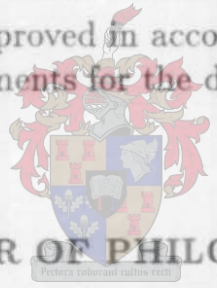
Training, Dynamics, and Complexity of Architecture-specific Recurrent Neural Networks

by

Jacques Ludik

Dissertation approved in accordance with the requirements for the degree of



DOCTOR OF PHILOSOPHY

at the

University of Stellenbosch

Department of Computer Science

Promoter: *Professor I. Cloete*

December 1994

This dissertation is dedicated to
my wife, *Elna*,
my parents, *Hennie* and *Anita*,
and my brothers, *Hein* and *Erens*.

DECLARATION

I hereby certify that this dissertation is my own original work, except where specifically acknowledged in the text. Neither the present dissertation nor any part thereof, has previously been submitted at any other university for a degree.

J. LUDIK

December 1994

SUMMARY

This dissertation describes the main results of a pioneering effort to develop novel architectures, training strategies, dynamics analysis techniques and theoretical complexity results for architecture-specific recurrent neural networks (ASRNNs). To put the study of ASRNNs into an appropriate perspective, a temporal processing framework that describes the different neural network approaches taken, was constructed. ASRNNs are more powerful than non-recurrent networks and computationally less expensive, more stable, and easier to study than general-purpose recurrent networks. The focus was on Elman, Jordan, and Temporal Autoassociation ASRNNs using discrete-time backpropagation.

Training strategies have been overlooked as a possible method to speed up learning and address the defects of backpropagation. Nine new training strategies were developed to produce effective solutions within a feasible time for different kinds of ASRNNs and feedforward networks. It was shown that they outperform all the conventional training strategies, using a variety of applications, which vary in complexity and range from sequence recognition to sequence generation. Several ASRNN architectures were compared at the same time and one of our suggested ASRNN architectures, the Output-to-Hidden Hidden-to-Hidden ASRNN, was found to be the superior architecture, outperforming the conventional ASRNNs by between 44% and 87% for different training strategies. *Increased Complexity Training* was developed for applications where the training set can be partitioned into subsets of increasing complexity. *Incremental Subset Training* and *Incremental Increased Complexity Training*, which include an incremental schedule to algorithmically determine RMS termination values on each subsequent subset, were also proposed. For the correct increment in subset size the number of updates required for the addition task almost attained the theoretical lower bound. *Smallest Delta Subset Training*, *Largest Delta Subset Training*, *Alternating Delta Subset Training*, and their incremental versions were also suggested. These strategies determine the complexity relation between the input patterns by obtaining their inter-pattern distances and then ranking the patterns according to some scheme. The incremental Delta training strategies performed the best overall. By visualizing the movement of hyperplanes and weights, it was shown how the new training strategies efficiently reduce the weight activity and obtain faster convergence.

The classification capabilities and dynamics of the Elman, Jordan, and Temporal Autoassociation ASRNNs were investigated. For the theoretical analysis, the threshold versions of these networks were considered. For each network the possible types and number of cells in the input and hidden unit activation space were determined. Account was given of the effect of adding context and state units by comparing feedforward networks with their corresponding ASRNNs in terms of various examples and interpreting the equations for the number of cells. The dynamics of the classification process was further explored by analyzing the clusters formed by

the hidden unit activations of the network. The clusters were obtained by visualizing the input activation space and using Hierarchical Cluster Analysis, Principal Component Analysis, and Sammon Transformation Analysis. It was shown that an Elman ASRNN can learn to simulate a finite state machine. The dynamics of the learning process in ASRNNs were explored by visualizing the hidden activation time traces and the evolution of clusters formed during learning. It was also shown that the network's performance improves with longer hidden unit activation histories.

The complexity analysis of Elman, Jordan, and Temporal Autoassociation threshold networks was continued by proving upper and lower bounds for their capacity and number of hidden units. The capacity of the networks was addressed in terms of the number of examples in general position and the Vapnik-Chervonenkis dimension. Since the number of hidden units is in practice chosen experimentally, bounds for the number of hidden units is of particular importance for the optimal design of an ASRNN architecture. In addition we have corrected some mistakes previously made in the complexity analysis of feedforward networks. The conclusions of this analysis can be extended to ASRNNs with the sigmoid activation function.

In this study promising techniques were developed and results obtained for architectures, training strategies, dynamics analysis, and complexity analysis of architecture-specific recurrent neural networks.

OPSOMMING

Hierdie proefskrif beskryf die belangrikste resultate van baanbrekerswerk om nuwe argitekture, afrigstrategieë, dinamiese analise tegnieke en teoretiese kompleksiteitsresultate vir argitektuur-spesifieke terugvoer neurale netwerke (ASTNNe) te ontwikkel. Om die studie van ASTNNe in 'n gepaste perspektief te plaas, is 'n temporale verwerkingsraamwerk daargestel wat die verskillende neurale netwerk benaderings wat ingespan word, beskryf. ASTNNe is kragtiger as nie-terugvoer netwerke en minder berekeningsintensief, meer stabiel, en eenvoudiger om te bestudeer as meer-doelige terugvoer netwerke. In hierdie studie word spesifiek gefokus op Elman, Jordan, en Temporale Outoassosiasie ASTNNe wat van die terugpropagering leer-algoritme gebruik maak.

Afrigstrategieë is afgeskeep as 'n moontlike metode om die leerproses te versnel en tekortkominge van terugpropagering aan te spreek. Nege nuwe afrigstrategieë is ontwikkel om effektiewe oplossings binne 'n aanvaarbare tyd te produseer vir verskillende tipes ASTNNe en vorentoe-voer netwerke. Daar is aangetoon dat hulle al die konvensionele afrigstrategieë oortref vir 'n verskeidenheid van toepassings wat varieer in kompleksiteit en strek van reeksherkenning tot reeksgenerasie. Verskillende ASTNN argitekture is terselfdertyd vergelyk, en een van ons voorgestelde ASTNN argitekture, die Afvoer-na-Verborge Verborge-na-Verborge ASTNN, is onderskei as die voortreflikste argitektuur. Die betrokke ASTNN het die ander konvensionele ASTNNe met tussen 44% en 87% oortref vir verskillende afrigstrategieë. *Toenemende Kompleksiteitsafrigting* is ontwikkel vir toepassings waar die afrigversameling verdeel kan word in subversamelings van toenemende kompleksiteit. *Inkrementele Subversamelingsafrigting* en *Inkrementele Toenemende Kompleksiteitsafrigting*, wat 'n inkrementele skedule insluit wat die fout termineringswaardes vir elke opeenvolgende subversameling algoritmies bereken, is ook voorgestel. Vir die korrekte inkrement in subversamelinggroote was die aantal gewigbywerkings wat vir die optellingstaak benodig word, byna gelyk aan die teoretiese ondergrens. *Kleinste Delta Subversamelingsafrigting*, *Grootste Delta Subversamelingsafrigting*, *Afwisselende Delta Subversamelingsafrigting*, en hul inkrementele weergawes is ook ontwikkel. Hierdie strategieë bepaal die kompleksiteitsverhouding tussen toevoerpatrone deur hul interpatroon afstande te bereken en die toevoerpatrone dan te rangskik volgens 'n spesifieke skema. Die inkrementele Delta afrigstrategieë het in die geheel die beste vertoon. Die visualisering van die hipervlakke en gewigwaardes se beweging het aangetoon hoe die nuwe afrigstrategieë die gewigsaktiwiteit effektief verminder om vinniger konvergensie te bewerkstellig.

Die klassifikasie vermoëns en dinamika van die Elman, Jordan, en Temporale Outoassosiasie ASTNNe is voorts ondersoek. Vir die teoretiese analise is die drempelweergawes van hierdie netwerke beskou. Vir elke netwerk is die moontlike tipe en aantal selle in die toevoer en verborge aktiveringsruimte bepaal. Daar is ook rekenskap gegee van wat plaasvind indien die konteks- en toestandseenhede vermeerder word deur die vorentoe-voer netwerke met hul ooreenstemmende

ASTNNe te vergelyk in terme van voorbeelde en interpretasies van die vergelykings vir die aantal selle. Die dinamika van die klassifikasie proses is verder ondersoek deur die groeperings te analiseer wat deur die verborge eenhede se aktiverings gevorm word. Die groeperings is bepaal deur die toevoer aktiveringsruimte te visualiseer en gebruik te maak van Hierargiese Groeperingsanalise, Hoofkomponentanalise, en Sammon Transformasie analise. Daar is aangetoon dat 'n Elman ASTNN kan leer om 'n eindige toestand outomaat te simuleer. Die dinamika van die leerproses in ASTNNe is ook ondersoek deur die visualisering van die verborge eenhede se aktiveringstydspore en die evolusie van groeperings wat gevorm word gedurende die leerproses. Daar is ook getoon dat die netwerk se werkverrigting verbeter met 'n langer verborge eenheidsgeskiedenis.

Die kompleksiteit van Elman, Jordan, en Temporale Outoassosiasie drempel netwerke is verder geanaliseer deur bo- en ondergrense vir die kapasiteit en aantal verborge eenhede te bewys. Die kapasiteit van die netwerke is aangespreek in terme van die aantal voorbeelde in algemene posisie en die Vapnik-Chervonenkis dimensie. Aangesien die aantal verborge eenhede in die praktyk eksperimenteel bepaal word, is grense vir die aantal verborge eenhede van besondere belang vir die optimale ontwerp van 'n ASTNN argitektuur. Verder is daar ook foute gekorrigeer wat voorheen gemaak is in die kompleksiteitsanalise van vorentoe-voer netwerke. Die gevolgtrekkings en resultate van hierdie analise kan uitgebrei word na ASTNNe met eenhede wat die sigmoidale aktiveringsfunksie implementeer.

In hierdie studie is belowende tegnieke ontwikkel en resultate verkry ten opsigte van die argitekture, afrigstrategieë, dinamiese analise, en kompleksiteitsanalise van argitektuur-spesifieke terugvoer neurale netwerke.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to our Lord Jesus Christ, for all his wonderful blessings, directed guidance, and tangible support throughout my studies, which culminated in the completion of this dissertation. The main purpose of this dissertation is to glorify Him.

The study was made possible by the contributions and true support of many people.

- I would like to extend my sincerest appreciation to my promoter, Professor Ian Cloete, for the guidance that he has given over the past years, his steadfast motivation at times when it was needed most, and his unwavering support throughout the study period. I would also like to thank his wife, Wilma Cloete, for her support.
- I am also indebted to all my colleagues and friends at the Department of Computer Science and Engineering. My gratitude is especially due to my colleagues, in particular, Professor Tony Krzesinski, for giving me the opportunity to complete the Ph.D., whilst lecturing. I would also like to thank, in alphabetical order, Pieter de Villiers, Reg Dodds, Dr Etienne Gouws, Harry and Lynette Lewis, Deon Mitton, Albriëtte Senekal and Herna Victor for their support and interest. My gratitude is also due to Etienne van der Poel, for the insightful discussions and help to produce some of the figures in this dissertation, Andries Engelbrecht, Pieter Theron, Hendrik Theron, Jacques Hayter, Jaco van der Merwe, Otto Strydom, Johan du Preez, Adri Conradie, Kobus Duvenhage and all the post graduate students in our department.
- I would also like to thank Prof Jacek Zurada, Dr C.H. Rohwer and Prof J.C. Olivier for their time and effort in examining the dissertation.
- To all my friends who supported me, I would like to express my deepest gratitude: In particular, Alten and Ilse du Plessis, Jurie and Sanet de Kock, Derik and Carin Mocke, Kobus and Anne-Marie Genis, Robert and Sonja Vosloo, Gerhard Cruywagen, Francois Geldenhuys, and Pierre Grigor.
- To my brothers, Hein and Erens, the following: Thank you for your tremendous support, love, and friendship. My gratitude is also due to my parents in law, Eric and Trudie Olivier, for their constant support and interest. I would also like to thank Hein's wife Gideoni, Albie and Lihani Olivier, Jannie Ludik, my grandparents Ludik and Retief, and other relatives for their support.
- I would like to express my deepest gratitude to my parents, Hennie and Anita, for their love, commitment, motivation, and support over many years. This dissertation is in particular dedicated to the memories of my father who passed away on the 27th of May 1994.

• To my wife, Elna, I cannot thank you enough for your love, tolerance, support, and wonderful commitment. Thank you so much for everything...

To the only God, who alone is all-wise,

be the glory through Jesus Christ for ever!

Romans 16:27

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Life is temporal. Many human behaviours express themselves as temporal sequences. Biological networks are known to exhibit time-variant behaviour. However, most artificial neural networks are not able to deal with temporal data and have focused on time-invariant pattern classification tasks. A variety of different neural network approaches, e.g. the *architecture-specific recurrent neural network* (ASRNN) approach, have been tried for storage, generation and recognition of temporal sequences. They have been applied to, among others, language processing, speech processing, control, signal processing, fault diagnosis, and the modelling of biological networks and psychological behaviour.

Two main types of temporal representations may be used to incorporate time in neural networks. These are *explicit spatial representations of time*, which map time into space (i.e. encoding the order of inputs by their spatial position in a network layer of units), and *dynamical representations of time*, which represent time implicitly by using networks with feedback connections. Neural networks dealing with temporal processing can be classified into recurrent networks (networks having feedback connections) and non-recurrent networks, generally referred to as feedforward networks. *Recurrent* networks use a dynamical representation of time and comprise general-purpose, architecture-specific, sequential associative, and psychophysiological networks. General-purpose recurrent networks use unlimited architectures and learning algorithms that can deal with time-varying input and/or output in nontrivial ways. For architecture-specific recurrent neural networks, also called simple recurrent networks, feedback connections are organized in strict architectures such as hidden-to-hidden, output-to-hidden, and output-to-output unit feedback. Sequential associative networks extend past work on associative memory to the case where pattern sequences can be stored and the recognition of a sequence corresponds to the

network settling into a desired sequence of stable states. The psychophysiological approach is characterized by networks that model physiological (biological) networks and/or psychological behaviour. The *non-recurrent* approach consists essentially of spatial representation time delay networks and other special constructed feedforward networks. Figure 1.1 illustrates the different temporal processing approaches in a hierarchical fashion.

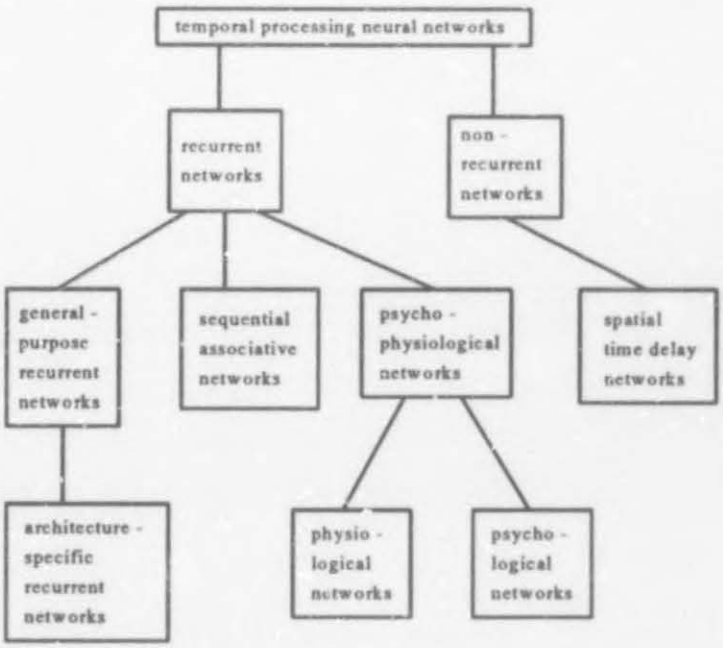


Figure 1.1: Temporal processing networks

Many researchers have focused their efforts in the last couple of years on recurrent neural networks because of their capability to exhibit dynamic behaviour in time. These networks, depending on their architecture and weights, can be trained to behave as associative memories [Pineda, 1987, Almeida, 1987], as oscillators [Pearlmutter, 1989], and also as finite state automata [Cleeremans *et al*, 1989, Williams & Zipser, 1989, Elman, 1990, Giles *et al*, 1992]. Although they represent a very powerful computational model, the design of proper architectures for a given problem and the devising of learning algorithms and training strategies is a very demanding task. In this dissertation we restrict our attention to architecture-specific recurrent neural networks, since they are more powerful than non-recurrent networks and computationally less expensive, more stable, and easier to study than general-purpose recurrent networks. In particular, we address the problem areas of these networks such as architectural design, training optimization, dynamics analysis, and capability complexity. In Chapter 2 we outline the development and current state of ASRNNs in the research field, and suggest some alternative architectures. The rationale of this study is firstly to develop *training strategies* to produce

effective solutions within a feasible time for different kinds of ASRNNs (Chapter 3). Training strategies have been overlooked as a possible method to speed up learning and address the problems of some learning algorithms. We show that the nine newly introduced training strategies outperform all the conventional ones, not only for different ASRNNs, but also for feedforward networks using a variety of applications. We further demonstrate the superiority of one of our suggested architectures with the training strategies. Secondly we analyze the *classification capabilities* and *dynamics of the classification and learning process* of ASRNNs in Chapter 4. It is also shown that an ASRNN can learn to mimic closely a finite state machine. Thirdly, in Chapter 5, the theoretical *complexity analysis* of architecture-specific recurrent threshold networks is further investigated by obtaining bounds for their capacity and number of hidden units. In addition we correct some mistakes previously made in the complexity analysis of feedforward networks. Finally, Chapter 6 is devoted to the summary of the main results, inferences, and suggestions for future research.

The first section of this chapter presents a formal description of a neural network, with special reference to a recurrent neural network and its dynamics. An overview of learning algorithms for training networks to perform temporal tasks is also given. To put the study of ASRNNs into an appropriate perspective, the temporal processing framework is further expanded in terms of the different kinds of recurrent and non-recurrent networks (sections 1.2 and 1.3). Sections 1.4 through 1.6 provide specific objectives and motivations for architectures, training strategies, classification and learning dynamics, and complexity issues of ASRNNs in the sequence of treatment in this dissertation.

1.1 Neural network formalization

Fiesler & Caulfield (1992) constructed a hierarchical description of a neural network that is split into a static and a dynamic part. The static part deals with network parameters that remain constant during learning and recall, and consists of the network's architecture (topology) and constraints. The constraints state the value ranges for the activations, local thresholds, and weights. The architecture comprises both the framework of a network (the number of layers and number of units per layer) and the interconnection scheme, which is determined by the following four properties:

- which units are connected (*connectivity*);

- the types of connections used: *intralayer* (between units of the same layer), *interlayer* (between units in adjacent layers), and *supralayer* (between units that are neither in adjacent layers, nor in the same layer);
- whether the connections are *symmetric* (connection having the same weight when used in either direction) or *asymmetric* (connection whose weight is only used for propagation in one direction);
- the *order* of the connections; a higher order (sigma-pi) connection is a connection that combines inputs from several units, usually by multiplication (the order is determined by the number of inputs).

The network dynamics can be depicted by the transition functions and initial state values for the activations, local thresholds and weights. The following transition functions can be used to determine the next states of the network:

- *Transfer functions*, which compute the activation of a unit, given its inputs. The transfer function is composed of two parts: a similarity measure and an activation function. The *similarity measure* measures the similarity between the input activity pattern and the weights connecting the unit to other units of the input activities. Two common similarity measures are Inner product and Euclidean distance (see Appendix A). The *activation function* maps the similarity measure to a prespecified activation range. Five common examples are the Linear, Ramp threshold, Step threshold, Gaussian and Sigmoid activation functions (see Appendix A).
- *Learning functions (algorithms)*, which specify how weights and thresholds will be updated. Two types of algorithms can be identified: *supervised* and *self-organized* (unsupervised). These will be discussed in the following section.
- *Clamping functions*, which determine if and when certain units retain their present activation value.
- *Ontogenic functions*, which define changes in the network architecture.

Network architectures may be divided into two principal types: *recurrent* and *non-recurrent networks*, i.e. feedforward networks. Feedback is permitted in a recurrent network (see Figure 1.2), but not in a non-recurrent one. *Feedforward* architectures are an important sub-class of non-recurrent networks in which units are organized into layers, and only asymmetrical connections

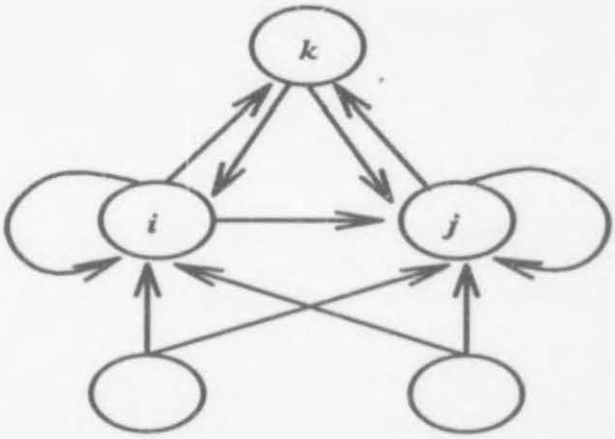


Figure 1.2: A recurrent neural network. Examples of feedback connections include those from units i and j to themselves, and those from unit k to units i and j.

are permitted between adjacent layers. In Figure 1.2, the unlabeled circles denote input units (of the input layer), the circles labeled i and j denote hidden units (of the hidden layer), and the circle labeled k denotes the output unit (of the output layer).

A more formal definition of a recurrent neural network is [Fiesler & Caulfield, 1992]: A *recurrent neural network* is either a symmetric neural network (has only symmetric connections) which contains at least one cycle (simple closed path) or an asymmetrical neural network (has one or more asymmetric connections) that contains at least one circuit (closed chain).

Neural networks can also be defined as nonlinear *dynamical systems* where network computation and learning can be described as processes moving in state spaces by constantly applying the transition functions. The state space defined by the input unit activations is called the *input activation space*, whereas the state space defined by the hidden unit activations is called the *hidden activation space*. The state space defined by all the weights in the network is referred to as the *weight space*. The learning process is a search in weight space in order to obtain the desired network computation. Recurrent networks can map all inputs to *fixed points* (equilibrium points), which act as steady state attractors, or desired *trajectories*, which act as limit cycles (two or more unique sets of activations being repeatedly visited) or non-point attractors such as periodic attractors. The set of points in the state space that is attracted by the fixed point or trajectory is called the *attractor basin*. *Global stability* is the eventual stabilization of all the unit activations in the network from any initial input. Equilibria and trajectories that are extensively disturbed by deviations are termed unstable. The stability at a fixed point or a trajectory can be proven by using a Lyapunov energy function [Cohen & Grossberg, 1983]. In the case of a

trajectory the Lyapunov function can be constructed from the entire sequence of input patterns.

Currently there exists two general types of algorithms used to train networks to perform temporal tasks, namely *supervised* and *self-organized* (unsupervised). Supervised algorithms use an external teacher and/or global information, while self-organized algorithms only relies on internal control and local information. Two examples of a temporal supervised learning task are sequence classification, where the input is a time-varying sequence and the desired output is the correct classification, and sequence production, where the input is a constant pattern and the desired output is a time-varying sequence. In general both the input and desired output may be time-varying. Although temporal self-organized algorithms are more complex than supervised learning algorithms, they are biologically more plausible.

In the following two sections different supervised and self-organized learning algorithms are briefly discussed. The algorithms used in temporal processing networks are either the same ones used by their static counterparts or otherwise extensions of them. The most prominent types of learning algorithms are captured in Figure 1.3.

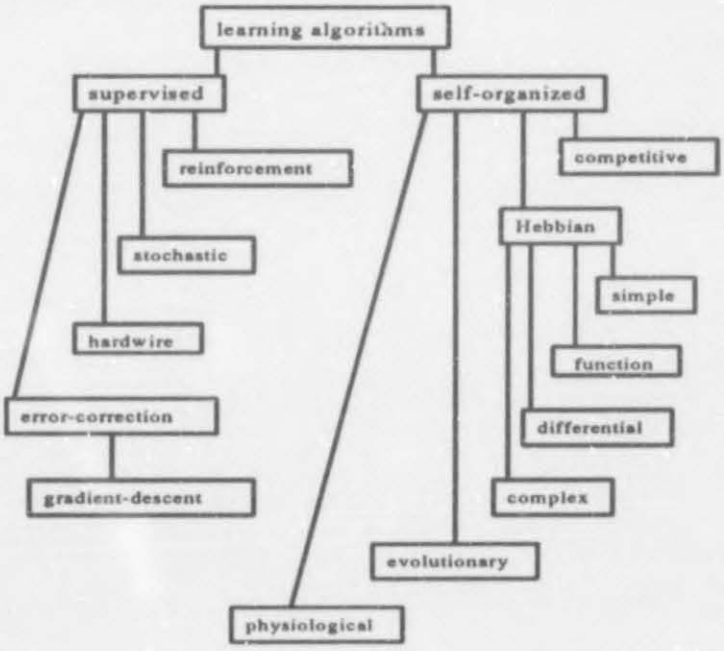


Figure 1.3: Learning algorithms

1.1.1 Supervised learning

Examples of supervised learning include error-correction learning, stochastic learning, reinforcement learning, and hardwire learning.

In *error-correction learning* the weights are changed according to some performance measure of the error between the desired and computed activation values of each output unit. The most prominent type of error-correction algorithm is a *gradient-based learning* algorithm, where at least part of the algorithm involves computing the gradient of the performance measure for the network in weight space or even activation space [Rohwer, 1989]. The gradient can be computed either exactly or approximately, the latter having more desirable computational features but being less accurate. The popular backpropagation (BP) algorithm [Rumelhart *et al*, 1986b] is an example of a gradient-based learning algorithm that computes the error gradient in weight space. In this study we focus on BP learning in ASRNNs. Other examples of neural networks that use an error-correction algorithm include the Perceptron [Rosenblatt, 1957], Adaline/Madaline [Widrow, 1962], and the recurrent Brain-State-in-a-Box [Anderson, 1977].

In *stochastic learning* weights are randomly changed using an energy relationship, a probability distribution, and a process that uses noise to escape local energy minima (simulated annealing). The change is unconditionally accepted if the energy is lower after the random weight change, and accepted according to a predetermined probability distribution if the change is higher. The weight change is rejected if it cannot be accepted according to the probability distribution. Examples of networks using stochastic learning include the Boltzmann Machine [Hinton *et al*, 1984] as well as its recurrent version [Prager *et al*, 1986], and Cauchy Machine [Szu, 1986].

Reinforcement learning differs from error-correction learning in that there is only one reward/punish value for all the output units, while in the latter case their is a error value for each output unit. This learning also concerns when corrections must be made (temporal corrections), as opposed to where corrections must be made (structural corrections) to improve the network performance. Reinforcement learning occupies, in some sense, a middle ground between supervised and self-organized learning, because the temporal corrections are made from a limited amount of information. Examples of networks using this type of learning include the Adaptive Heuristic Critic [Barto *et al*, 1983], Associative Reward-Penalty [Barto & Anandan, 1985], and Temporal-Difference [Sutton, 1987].

In *hardwire learning* the connections and weights of the neural network are predetermined. The supervised version of the recurrent Fuzzy Cognitive Map [Kosko, 1986] is an example of a

network where each unit represents a concept and causal signed values are assigned (hardwired) to each connection.

1.1.2 Self-organized learning

Self-organized learning consists essentially of Hebbian and competitive learning. The *Hebbian learning* rule [Hebb, 1949] states briefly that if two units are activated at the same time, the weight between them must be increased. In *competitive learning*, also called "winner take all" or "on-center/off-surround" learning, each unit competes with other units by sending positive signals to itself (self-excitation via a feedback connection) and negative signals to all its neighbouring units. An extension to competitive learning is competitive-cooperative learning where some of the neighbouring units also receive positive signals.

In the simplest form of Hebbian learning, *simple Hebbian learning*, the weight change is the correlation of the activations of the two units involved. Examples of recurrent networks using simple Hebbian learning with feedback recall include the Discrete Autocorrelator [Amari, 1972, Hopfield, 1982], Discrete Bidirectional Associative Memory [Kosko, 1988a], and Temporal Associative Memory (TAM) [Amari, 1972].

In *function Hebbian learning* (passive decay learning) the weight change is the correlation between the two activation values after they have been passed through a nonlinear activation function. Examples of recurrent networks using function Hebbian learning with feedback recall include Additive Grossberg [Grossberg, 1968], Shunting Grossberg [Grossberg, 1973], Continuous Hopfield [Hopfield, 1984], and Adaptive Bidirectional Associative Memory (ABAM) [Kosko, 1987a].

Differential Hebbian learning correlates the time derivatives of the nonlinear activation functions. Examples of networks using differential Hebbian Learning include Drive-Reinforcement [Klopf, 1987], the recurrent Fuzzy Cognitive Map [Kosko, 1986], and the recurrent Differential Hebbian ABAM [Kosko, 1988b].

Examples of networks using *competitive learning* (gated decay learning) include Self-organizing Feature Maps (Learning Vector Quantization) [Kohonen, 1988], Adaptive Resonance Theory (ART) [Carpenter & Grossberg, 1986].

There is a final stream of self-organized learning which regards the above-mentioned approaches as too simple and advocate that they should be physiologically-based and extended towards more

complex Hebbian [Von der Malsburg, 1973] [Linsker, 1988] and evolutionary models (proposed by Edelman (1988) and Goldberg (1989)). In the more *complex Hebbian models* of Von der Malsberg and Linsker the learning rule adapts all connections in all directions. In biological systems, some self-organization occurs during the life of an organism, and some over evolutionary time. *Evolutionary models*, which are based on Darwin's selection theories and genetic algorithms (successful populations multiply faster than ones that are unsuccessful), can implement population theories in neural networks. Physiologically-based models incorporate units that approximate biological neurons more closely such as *leaky integrators*, and transmit *pulse-coded* information. Reiss & Taylor (1991) presented a network composed of leaky integrator neurons for the direct storage of temporal sequences.

1.2 Recurrent networks

Recurrent neural networks can perform highly nonlinear dynamic mappings and thus temporally extended applications, whereas multilayer feedforward networks are confined to performing static mappings. The fact that biological neural networks are highly recurrently connected also support the use of such networks. While time delay feedforward networks can only manage limited forms of time-varying behaviours, recurrent networks manifest not only a special type of behaviour such as settling to a fixed stable state (associative memories), but a much greater variety of dynamical time-varying behaviours and the ability to deal with them through their own natural temporal operation.

Offline and *online* approaches can be distinguished in training recurrent networks. In feedforward networks their equivalent are respectively called *batch* approaches, in which weight changes are performed at the end of each epoch (presentation of all the patterns), and *incremental* or *stochastic* approaches, in which weight changes are made after each pattern. A single epoch for a recurrent network corresponds to one training pattern for a feedforward network. In offline (epochwise operating or lab-time) learning a network is reset to its starting state after each epoch and weight updates are made only at epoch boundaries, but in online (continually operating or real-time) learning no state reset occurs and weight updates can be performed at all time steps. In offline learning a network can be trained in a batch or incremental fashion.

Continuous-time recurrent networks are governed by coupled differential equations using relaxation time constants, whereas *discrete-time* recurrent networks use the first order finite difference approximation of these equations. A continuous-time network equation can be transferred to a

discrete-time equation by setting the time constant equal to 1 and substituting $y(t+1) - y(t)$ for $dy(t)/dt$. However, the behaviour of biological neural networks can more realistically be simulated by continuous-time networks than by discrete-time networks.

Recurrent neural networks also have important capabilities such as attractor dynamics and the ability to store information for later use, which are not found in feedforward networks. There is an important difference between gradient descent learning in feedforward and recurrent networks: if each unit in a feedforward network has a smooth transfer function, the output of the network is a continuous function of the weights; in a recurrent network the output can change greatly with a very small change in the network parameter when it passes through a bifurcation point (a critical parameter value where the attractor changes). This means that a gradient descent algorithm used in a recurrent network may not converge, because the error surface of such a network may be discontinuous in weight space [Doya, 1992].

Problems with bifurcations of gradient descent learning for recurrent networks include [Doya, 1992]:

- *Instability of learning dynamics*: When an equilibrium bifurcates, the asymptotic stability of the learning equation cannot be guarenteed. When an equilibrium changes into a limit cycle through a Hopf bifurcation, the solution of the learning equation grows exponentially and causes the weights to change drastically.

- *Non-convergence of learning*: The error can increase as learning proceeds when the network's trajectory changes discontinuously with small parameter change in bifurcation points such as saddle-node and Hopf bifurcations. When a non-adaptive learning rate is used, long and random jumps in the parameter space can be caused by a very steep error gradient near bifurcation points.

- *Confusion of input-output patterns*: A network state space with too few attractor basins can find it difficult to discriminate input sequences and as a result only learns average output values. If the network starts with only one attractor basin, the network must endure some bifurcations, which may lead to some problems in the gradient descent learning.

Some solutions to overcome bifurcation problems in gradient descent learning include the following [Doya, 1992]:

- *Approximate learning algorithms*: By omitting part of the computation required to fully compute the exact error gradient, the instability of the learning dynamics can be avoided.

Such approximations were successfully used in ASRNNs for sequence prediction [Elman, 1990] and sequence production applications [Jordan, 1986].

- *Teacher forcing* is a technique where the actual output of the network is replaced by the desired output [Pineda, 1988] [Doya & Yoshizawa, 1989] [Williams & Zipser, 1989]: It can be used to synchronize the network dynamics to the desired output signal. Not all the output units have to be teacher-forced. A stable solution of a weaker teacher-forced network, where the actual output is replaced by a linear combination of the desired and the actual output, is more likely to remain stable.

- *Preprogramming of network structures*: The instability of the learning dynamics can also be overcome in situations where the required structure of the state and weight space are known beforehand. In such a case the network states and weights can be preprogrammed to have a certain amount of attractor basins.

Four types of recurrent networks are next discussed: architecture-specific, general-purpose, sequential associative, and psychophysiological networks. Since the architecture-specific recurrent network is a subclass of the general-purpose recurrent network, the latter is also of particular interest to this study. A more detailed review of the different recurrent approaches as presented in the temporal processing framework appears in [Ludik, 1992].

1.2.1 Architecture-specific Recurrent Neural Networks

Architecture-specific recurrent neural networks (ASRNNs), also called Simple Recurrent Networks or "partially recurrent" networks, involves treating a neural network as a simple dynamical system in which previous states are made available as an additional input to any layer, i.e. feedback connections are organized in strict architectures such as hidden-to-hidden, output-to-hidden, and output-to-output unit feedback. An example of an ASRNN with feedback connections from the hidden units back to themselves via an additional layer of units is illustrated in Figure 1.4. This network is called an *Elman ASRNN* [Elman, 1988] with one output unit and two hidden units having sigmoid activation functions, and two linear input units (units with linear activation functions) as well as two additional linear units (called context units), which contain a copy of the previous hidden activation values. The dotted arrows denote feedback connections with fixed weight values (typically, values of 1), whereas the solid arrows indicate feedforward connections with variable weight values. For ASRNNs with discrete-time dynamics, it is assumed that feedforward connections have delay 0 and the feedback connections have

delay 1, unless otherwise stated. Since the weights of the feedback connections are mostly fixed and information processing is sequential in time, training is essentially not much more difficult than for a standard backpropagation network (assuming BP is employed for the ASRNN). In Chapter 2 we outline the development and current state of the art of architecture-specific recurrent networks, discuss the benefits and limitations of ASRNNs, and suggest some alternative architectures. In subsequent chapters these recurrent networks are further investigated by developing training strategies for them to speed up learning and address learning problems such as instability and non-convergence, analyzing their dynamics during the learning and classification process, and exploring their complexity in terms of classification capabilities, capacity, and the number of hidden units.

In the next section we give examples of general-purpose recurrent neural networks and their learning rules, which can also be employed in ASRNNs, and conclude with their benefits and drawbacks.

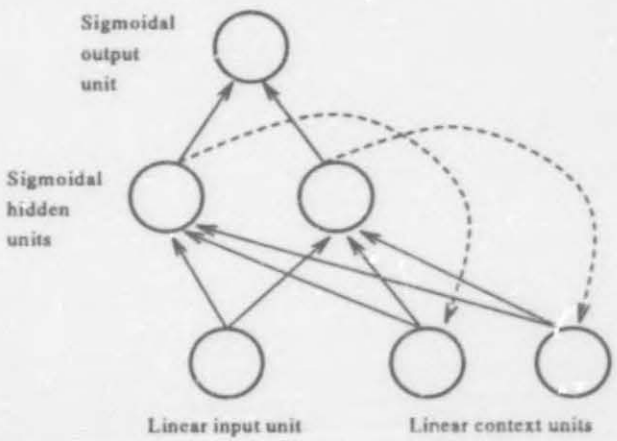


Figure 1.4: An ASRNN with hidden-to-hidden unit feedback connections

1.2.2 General-purpose Recurrent Neural Networks

General-purpose recurrent networks use unlimited architectures and learning algorithms that can deal with time-varying input and/or output in nontrivial ways.

Rumelhart *et al* (1986b) have developed the *backpropagation-through-time* (BPTT) approach, which can be derived by unfolding the temporal operation of a network into a multilayer feedforward network that grows by one layer on each time step. Four versions of the BPTT algorithm were described in [Williams & Zipser, 1990a, Williams & Peng, 1990b] (see Appendix A):

- *Epochwise BPTT* - the training stream is segmented into epochs; it saves the entire history of inputs, states and targets over an epoch-interval, and computes the exact error gradient;
- *Real-time BPTT* - the weight changes at each time step while the network continues to run; it saves the entire history of inputs and states, and also computes the exact error gradient in weight space;
- *Truncated BPTT* - a bounded-history approximation to real-time BPTT in which relevant information is saved for a fixed number k of time steps and any information older forgotten; it is denoted by BPTT(k) and computes an approximation of the error gradient;
- *Epoch-truncated BPTT* - it is obtained by combining aspects of epochwise and truncated BPTT; the network is run through k_1 additional time steps before the next BPTT computation, where $k_1 \leq k$; it approximates the error gradient in the weight space by taking into account only the $[t - k_1, t]$ part of the history over the interval $[t - k, t]$; it is denoted by BPTT($k; k_1$); it is evident that BPTT(k) is the same as BPTT($k; 1$), and BPTT($k; k$) is the epochwise BPTT algorithm;

Williams & Zipser (1989) have developed the *real-time recurrent learning* (RTRL) approach (see Appendix A), which enjoys the generality of the BPTT approach while not suffering from its growing memory requirement in arbitrarily long training sequences. Although the RTRL algorithm has great power and generality, it has the disadvantage of being computationally very expensive. Zipser (1989) described a technique, called *subgrouping RTRL*, for reducing the amount of computation required by RTRL without changing the connectivity of the networks. This is accomplished by dividing the original network into subnetworks for the purpose of error propagation while leaving them undivided for activity propagation. For a 12-unit network that learns to be the finite-state part of a Turing machine, a speedup of ten is reported using the subgrouping version of RTRL as opposed to the standard one. Williams & Zipser (1990a) and Schmidhuber (1992b) independently proposed a discrete-time hybrid Forward Propagation BPTT (FP-BPTT) algorithm, which combines aspects of both the forward propagation used in RTRL and the backward error propagation used in the BPTT algorithm (see Appendix A). It also computes the exact error gradient more efficiently than any other online algorithm currently available. See Table A.1 (Appendix A) for a comparison of these general purpose recurrent learning algorithms in terms of worst case storage and time complexities, exact or approximate gradient, and online versus offline computation.

The benefits of general-purpose recurrent networks include their generality, unlimited architectures, inherent parallelism, and flexible dynamics. Most of their algorithms also compute the

error gradient exactly, and BPTT has potentially a perfect memory. On the other hand, these networks also have basic limitations, such as the unlimited amount of memory that is demanded in each unit (for BPTT, there is a growing memory requirement when given an arbitrarily long training sequence), they scale up very badly and are computationally very expensive, they have slow learning convergence and do often not converge (unstable), they have a too complex local interconnection structure, and are biological unplausible (since all their algorithms are variations of backpropagation). With the ASRNN approach most of these drawbacks can be overcome, as indicated in Chapter 2.

The sequential associative recurrent approach, which is specially suited for learning sequences of stable states, is next briefly reviewed as an alternative to ASRNN for temporal processing.

1.2.3 Sequential Associative Neural Networks

Sequential associative networks, also called temporal spin-glass networks or sequential associative memories, use unrestricted architectures wherein recognition of an input temporal pattern sequence corresponds to the network settling into a desired sequence of stable states. These networks extend past work on symmetric associative memory networks by Hopfield (1982, 1984) to the case where pattern sequences instead of static patterns can be stored. In the absence of time delay a symmetric network will always converge to a fixed point attractor. Symmetric networks using the recurrent backpropagation algorithm (RBP) [Almeida, 1987, Pineda, 1987] also correspond to the Hopfield network. The RBP algorithm is related to the general-purpose recurrent BPTT and RTRL algorithms, but can be viewed essentially as a computationally attractive special case of BPTT appropriate for situations when both the computed and desired trajectories consist of settling to a stable state.

Several sequential associative networks were developed which use asymmetric connections and time delay to produce associative storage, recognition and recall of sequences. Kleinfield (1986), Sompolinsky & Kanter (1986), and Amit (1988) each developed sequential associative networks that use discrete-time Hebbian learning, with long and short delays on connections, to generate and recognize pattern sequences. Rohwer & Forrest (1987) presented a discrete-time BPTT algorithm that settles to stable states. Buhmann & Schulten (1988) proposed a Hopfield network with asymmetric connections and probabilistic units, using discrete-time Hebbian learning. Marcus *et al* (1991) have studied the dynamics of symmetric and asymmetric Hopfield networks with time delay using continuous-time Hebbian learning. They have showed how time delay in the response of network units can produce sustained limit cycle behaviour (oscillation) and

chaos.

The main advantages of sequential associative networks include their ability to settle into a desired sequence of stable states, and their neurophysiological justified units (by being probabilistic or having time delay built into them). These networks are also relative easy to study, since fixed-point dynamics are well understood and the use of the Lyapunov energy functional allows a stability analysis and a quantitative study of the asymptotic network behaviour. On the other hand, these networks have basic limitations such as their low power of generalization, their symmetric connections which are biological unrealistic, and their inability to deal with perceptual invariances (when compared to ASRNNs). ASRNNs also uses a less complex unit structure and are not limited to the special type of behaviour such as settling to fixed stable sequences.

The psychophysiological approach, which is the next class of recurrent networks to be briefly discussed, uses an even more complex unit and interconnection structure.

1.2.4 Psychophysiological Neural Networks

Psychophysiological temporal processing networks are characterized by networks that model physiological (biological) networks and/or psychological behaviour. Physiological networks are primarily concerned with the direct physiological implementation of an architecture and learning algorithm that can be mapped onto a known neurophysiological network. Psychological networks are, on the other hand, concerned by fitting models of biological neural networks to psychological data. Since the two approaches have much in common, it is sometimes difficult to classify networks that are motivated by psychophysiological results.

All detailed temporal modelling of biological neurons uses the *leaky integrator* (LI) model [Hodgkin & Huxley, 1952]. Neurons act as leaky integrators by storing incoming signals in previous fractions of a second on their surface. Reiss & Taylor (1991) proposed the Simple LI, Spin-glass LI, and Channel LI models. Of the many different LI models, ranging from the simple LI model to the *compartmental model* (where a neuron is divided in small compartments) [Hindmarsh & Rose, 1984], Reiss & Taylor (1991) favoured the Channel LI model. Examples of psychological temporal processing networks in the areas of speech perception include the COHORT model [Marslen-Wilson, 1987], the Interactive Activation and TRACE models [Elman *et al*, 1986] [Rumelhart *et al*, 1986b], and Node Structure Theory [MacKay, 1987].

The advantages of psychophysiological networks include their physiological plausibility, correspondence to psychological data, and usefulness for understanding the behaviour of biological neural networks. On the other hand these networks are computationally expensive and have a very complex local interconnection and unit structure.

1.3 Non-recurrent Neural Networks

In this section the temporal processing framework is further expanded in terms of the different kinds of non-recurrent networks. The non-recurrent approach constitutes spatial representation time delay networks and other special constructed feedforward architectures and learning algorithms for temporal processing.

The *time delay neural network* (TDNN), also called the "moving window" network, is a multi-layer feedforward neural network which can be trained with consecutive time-delayed frames of the input data. All the temporal information is conveyed within a single frame of the window. By moving the window through the data, the network is trained to output the correct response. Weights in the initial layers are allowed by the time-delayed input frames to account for variations in the input data. Temporal structure in the TDNN is then represented at increasing levels of abstraction and duration in progression from the input to the output layer.

Waibel *et al* (1988) and Lang & Hinton (1988) proposed a multi-layer perceptron (MLP) TDNN using backpropagation to recognize voiced stops and to classify acoustically similar phonemes. Unnikrishnan *et al* (1988) developed a Time Concentration Network (TCN) that uses backpropagation with variable length delays to recognize digits. Watrous (1988) developed a Temporal Flow Structured MLP to recognize phonemes and words, whereas McDermott & Katagari (1988) used a time-delayed version of Kohonen's LVQ network to recognize voiced stops. An example of another non-recurrent type of network is the recursive auto-associative memory (RAAM) network [Pollack, 1990], which (using BP) automatically develops fixed-width recursive distributed representations of finite training sets of variable-sized data structures such as sequences and trees.

The main advantage of spatial transformation time delay networks is that they are computationally efficient. However, ASRNNs are much more powerful than TDNNs, since they overcome the latter's problems with an exponentially growing training set [Hoekstra & Kooijman, 1991], fixed time window, and inability to generalize between input positions. In addition to these

drawbacks, non-recurrent networks are also biologically unrealistic.

We subsequently give further motivation and objectives for studying training strategies, dynamics analysis and complexity issues of ASRNNs. The main achievements of the dissertation in each of these problem areas are briefly stated.

1.4 Training Strategies for ASRNNs

Backpropagation is probably the most widely used training [illegible] many neural network projects. However, at times it fails to converge or exhibit a long training time. Since BP is sensitive to initial conditions [Kolen & Pollack, 1991], it suggests that ASRNNs will be even more sensitive to initial conditions and thus more prone to non-convergence. For instance, Cottrell and Tsung (1991) report experiments on an addition task which could not be learned at all using the Elman ASRNN. Bianchini *et al* (1994) showed that examples of local minima for feedforward networks can be associated with similar ones in recurrent networks.

Usually, when training a neural network, the experimenter selects a fixed set of training patterns and a disjoint set of test patterns, where the assumption is that these sets are representative of the problem to be learned. Then the neural network is trained using the fixed set of training patterns until a success criterion is met. This may require repeated trials with varying initial parameters, since the network may get stuck in a local minimum. When the training set is large, this problem is more pronounced. After training the generalization performance of the network is determined on the test set. Various methods have been investigated to speed up learning using the backpropagation algorithm. These include adaptive learning rate adjustment [Silva & Almeida, 1990], use of a momentum term in weight updates [Rumelhart *et al*, 1986b], and second order derivatives [Fahlman, 1989]. To a large extent, however, *training strategies*, i.e. *different methods for presentation of examples in accordance with some performance measures*, have been neglected as a possible method to speed up learning and address the problems of BP. In chapter 3, we have introduced *increased complexity training*, two *incremental* training strategies and six *delta* training strategies, and showed that they do not only alleviate these problems, but also lead to considerable improvement in learning performance for different ASRNNs as well as feedforward networks. We have also used the newly proposed training strategies to compare different architecture-specific recurrent neural networks and found one of our suggested architectures, the Output-to-Hidden Hidden-to-Hidden ASRNN, to be the most efficient one. For a better grasp of the training strategies' operation, the movement of hyperplanes and weights

for each one is visualized and discussed. The visualization graphs showed how the new training strategies lessen the wandering about in the weight space and obtain faster convergence.

1.5 Dynamics of Classification and Learning of ASRNNs

In exploring the classification dynamics of the ASRNN, we use the concept of a finite state machine (FSM), which was originally studied in a formal manner in [McCullogh & Pitts, 1943]. Kleene (1956) considered regular expressions and modeled the neural networks of McColloch & Pitts by finite state automata, proving the equivalence of the two concepts. Minsky (1967) proved that every finite state machine is equivalent to, and can be simulated by, some neural network. However, identifying a FSM that is simulated by an ASRNN is in practice quite a problem, since ASRNNs are properly thought of as dynamical systems and are often high-dimensional systems, and consequently difficult to study using traditional techniques. In Chapter 4 it is shown that an ASRNN can learn to simulate closely a FSM, by first constructing a transition diagram for different temporal applications, and then for each application identifying a Moore machine that corresponds with the internal representations of the network. The internal representations of the ASRNNs are analyzed using familiar techniques such as Hierarchical Cluster Analysis and Principal Component Analysis (PCA), as well as an unfamiliar one called Sammon Transformation Analysis (STA) [Sammon, 1969]. The latter produced superior clustering results compared to PCA. We have also determined the correspondence between the simulated FSM and the one constructed for the Addition training data. The learning dynamics of ASRNNs are explored by discussing the role of the recurrent connections during learning and classification, and their relationship to feedforward networks. The evolution of clusters formed by STA is visualized and discussed in terms of a mapping of inspection points on the learning curve, simulation results at each point, and learning curve behaviour. The visualization of hidden activation time traces, enabled us to study the temporal paths formed in the hidden activation space during learning. By examining the ASRNN with different temporal window sizes, it was determined that the longer the hidden activation history, the better the network's performance.

1.6 Complexity Analysis of ASRNNs

Since the influential work of McCulloch & Pitts (1943), many researchers have studied the capability complexity of feedforward neural networks of linear threshold units. The complexity

analysis of neural networks is concerned with its computational and classification capabilities. At the heart of the computational capabilities are how many examples a network can remember or learn (i.e. its capacity), and how many hidden units are needed to realize an arbitrary mapping. Much remains to be discovered concerning optimal design of neural networks, and completely satisfactory solutions have not been offered for any of these problems. The classification capabilities are determined by the possible types and number of cells in the input and hidden space of a neural network. This is a difficult mathematical problem, since it is related to certain problems of combinatorics and convex set theory. Many researchers have been struggling to find good estimates of these capabilities for feedforward networks. However, in spite of the importance of these problems, the complexity issues of architecture-specific recurrent threshold networks (ASRTNs) have not yet been analyzed. In Chapters 4 and 5, theoretical results are obtained that encompasses the computational and classification capabilities of these type of recurrent networks. We have examined the classification behaviour of Elman, Jordan and Temporal Autoassociation (TA) ASRTNs in terms of the possible types and number of cells the network is capable of forming in the input and hidden spaces. We have discovered, for example, that the Elman and TA ASRTNs do not have any imaginary or closed cells and are not capable of forming disconnected decision regions. Theorems are also proven to obtain upper and lower bounds for the capacity and the number of hidden units of these networks. A summary of the bounds obtained appears at the end of Chapter 5. In addition we have corrected some mistakes previously made in determining bounds for the number of hidden units and capacity of feedforward networks. The conclusions of the complexity analysis improve comprehension of what can be performed by different architecture-specific recurrent threshold networks, and can be extended to networks with other nonlinear activation functions.

The next chapter will outline the development and current state of the art of architecture-specific recurrent networks, discuss the benefits and drawbacks of ASRNNs, suggest some alternative architectures, and describe contributions of this study.

Chapter 2

Architecture-specific Recurrent Networks

The overall objective of this chapter is to present a brief overview of the research literature of architecture-specific recurrent neural networks. We discuss not only aspects such as architectural design, learning algorithms, and applications, but also training strategies and dynamics analysis when applicable. The benefits and limitations of ASRNNs in general are also given. We further suggest a couple of promising architectural designs for ASRNNs, some of which we test with various training strategies to be introduced in Chapter 3. We conclude with remarks about contributions of this study to the state of the art.

2.1 Development and Current State of the Art

In this study the Jordan and Elman ASRNN using the BP learning algorithm are of particular importance, since they are the most widely used ASRNNs and are suited to perform both sequence generation and sequence prediction/recognition. The development and current state of the art are firstly discussed for Jordan and related ASRNNs, then Elman and related ASRNNs, and finally other ASRNNs (in chronological order for each).

Jordan and related ASRNNs:

The recent history of ASRNNs started with the work of Jordan (1986), who has extended a discrete-time one-hidden-layer BP network with an additional state layer. The sigmoidal output

layer (units with a sigmoidal activation function) of the *Jordan ASRNN* (see Figure 2.1) feeds back to activate the linear state layer, which together with the linear input layer, activates the sigmoidal hidden layer. The units in the state layer also have fixed self-looping connections to linearly average the output values. The fixed, one-to-one recurrent connections (the dotted arrow from the output to the state units) give the network a memory so that a static input pattern can be associated with a sequentially ordered output pattern. Robinson & Fallside (1988a) showed how Jordan networks and state space equations in classical control theory are related, and Bourlard & Wellekens (1988) proved that these networks could be used to calculate local probabilities required in Hidden Markov Model recognizers. Anderson *et al* (1988) investigated Jordan networks with one or two hidden layers for a consonant and vowel recognition task, and found better results with two hidden layers.

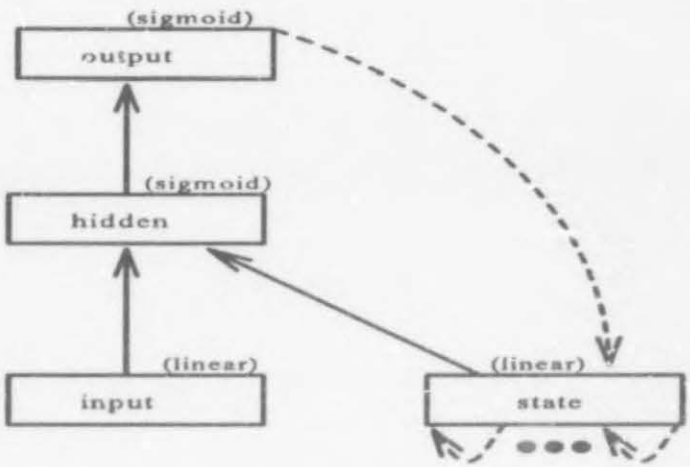


Figure 2.1: A Jordan ASRNN

We consider Jordan related ASRNNs to have feedback from the output to hidden layer via a state layer. See Table 2.1 for a summary of Jordan related ASRNNs. Prager *et al* (1986) explored the use of a Jordan-type of Boltzmann machine to speech recognition. Since the Boltzmann machine training algorithm is computationally more expensive than BP, the latter is preferred in this study.

Port & Anderson (1989) designed a Jordan ASRNN with sigma-pi connections from the state and input layer to the hidden layer. In a temporal application the network was trained by the discrete-time RTRL learning algorithm (see section 1.2.2) to recognize music tone sequences. This network exhibited limitations such as poor performance at variable input rate and weak representation of durational information. Williams & Zipser (1990a) also used a discrete-time hybrid FP-BPTT algorithm (see section 1.2.2) in a Jordan network. We avoid the instability

problems and computational expensiveness of these general-purpose learning algorithms, by using standard BP in Jordan ASRNNs.

Jain (1991) developed a modular recurrent neural network that learns to parse complex sentences presented one word at a time by acquiring a statistical grammar based on a combination of semantic and syntactic cues. The parser was constructed from three separately trained modules, each one similar to Jordan's approach. Two of the modules were constructed by a replication process that is similar to "weight sharing" in time delay neural networks, where equality constraints are placed on weights serving analogous functions in different input positions.

STUDY	ARCHITECTURE	LEARNING ALGORITHM
[Jordan, 1986]	Jordan network	Discrete-time BP
[Prager *et al*, 1986]	Jordan-type Boltzmann Machine network	Discrete-time Boltzmann reinforcement
[Anderson *et al*, 1988]	Jordan network with 1 or 2 hidden layers	Discrete-time BP
[Port & Anderson, 1989]	Jordan network with sigma-pi hidden layer	Discrete-time RTRL
[Williams & Zipser, 1990a]	Jordan network	Discrete-time hybrid FP-BPTT
[Jain, 1991]	Modular Jordan/TDNN network	Discrete-time BP
[Cottrell & Tsung, 1991]	Jordan network	Discrete-time BP
[Gordon *et al*, 1991]	Jordan network	Discrete-time BP
[Scholtes, 1991a]	Two layer extension of LVQ with feedback from outputs to inputs	Discrete-time LVQ
[Diederich, 1992]	Jordan network	Discrete-time BP with fixed or adaptable learning rates
[Imai *et al*, 1992]	Jordan network	Discrete-time BP
[Schulenburg, 1992]	Jordan network	Discrete-time BP
[Demura *et al*, 1993]	Jordan-like RFL network	Recurrent Flash Learning (RFL)

Table 2.1: Jordan and related ASRNNs (studies in chronological order)

Scholtes (1991a, 1991b, 1991c, 1991d) developed a two layer extension of the Learning Vector Quantizer (LVQ) (Kohonen feature map) network with feedback connections from the output

layer to the input layer. The first map categorizes single words, while the second map derives and categorizes contexts. This self-organizing recurrent ASRNN learned to derive simple semantics from unformatted sentences. A comparison between this network and the Elman ASRNN for the same applications reported in [Elman, 1988, Cleeremans *et al*, 1989] resulted in similar performances.

Imai *et al* (1992) trained a Jordan ASRNN to recognize printed sequential plural patterns independent of the order of the individual patterns. The network using discrete-time BP performed encouraging, even when the individual patterns were presented with a slight difference in position and size.

A Jordan network using discrete-time BP was trained by Schulenburg (1992) to process the words of sentences sequentially to produce case/role meaning representations. The recurrent feedback of this model is said to be "realistic" in natural language processing terms, since the meaning representation is built incrementally as opposed to training with the entire sentence meaning.

Demura *et al* (1993) proposed a learning algorithm for a Jordan-like ASRNN called *Recurrent Flash Learning* (RFL), which can model dynamical systems requiring only a single presentation of training sets for learning. The RFL algorithm requires each sigmoidal output unit of the ASRNN to be connected to a single linear unit, which in turn has a feedback connection to a self-looping linear unit. The latter unit has then connections to the linear state layer, preserving the network's history. The sigmoidal hidden units are constructed during the learning process.

Elman and related ASRNNs:

Elman (1988, 1989, 1990) has augmented a discrete-time one-hidden-layer BP network with an additional context layer. The sigmoidal hidden layer of the *Elman ASRNN* (see Figure 2.2) uses fixed one-to-one feedback connections (dotted arrow) to activate the linear context layer, which together with the linear input layer activate the hidden layer in the next time step. This architecture was applied to problems such as sequential XOR and the discovery of syntactic/semantic categories in natural language data. The BP algorithm used in the Elman network is equivalent to the truncated BPTT with history of one time step (BPTT(1)) in the recurrent portion and feedforward BP in the remainder of the network. Cottrell & Tsung (1991) pointed out that this approach is similar to having all the hidden units be completely interconnected and back propagating one time step along the recurrent connections. They compared Elman and Jordan ASRNNs on an addition task and concluded that the Elman network can learn constructions

that cannot be learned with the Jordan network, because networks with only output memory cannot remember things about their input that are not reflected in their output. Cottrell & Tsung also introduced a training strategy, called *Combined Subset Training* (see section 3.2 for a description and Chapter 3 for comparisons with strategies developed here), which showed improved performance when compared to training on a fixed set. They further used Principal Component Analysis to analyze the internal representations of the Elman ASRNN to get an idea of how the network makes transitions. Table 2.2 provides a summary of Elman related ASRNNs.

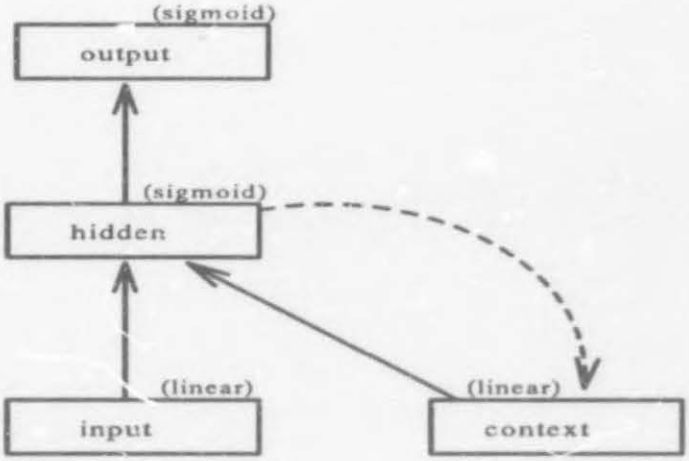


Figure 2.2: An Elman ASRNN

Servan-Schreiber *et al* (1988, 1989, 1991) and Cleeremans *et al* (1989) used an Elman ASRNN to train an infinite corpus of strings from the Reber finite state grammar with a discrete-time BP algorithm. They have used Hierarchical Cluster Analysis to analyze the internal representations and conclude that the network develops internal representations that correspond to the units of the grammar and closely approximates the corresponding minimal finite state recognizer.

Gordon *et al* (1991) compared the effectiveness of the Elman, Jordan and feedforward BP networks at predicting trajectories. The recurrent networks provided superior results. The Jordan network predicted the trajectory sooner and also indicated that a fully recurrent network might provide even better performance.

Iooss (1991) extended the Elman ASRNN for lexical decoding in a continuous speech recognition application. Self-looping connections on slow-decaying linear context units were added to prevent information about sequential context to be lost when errors occur in the input sequence. The sigmoidal output units in this network were also used to label patterns, as opposed to their prediction function in standard Elman ASRNNs.

Bartell & Cottrell (1991) developed a network, called Movie Description Network (MDN), which learns via discrete-time BP to generate sequences of words classifying sequences of perceptual input from an analog temporal environment. The MDN architecture consists of two coupled Elman networks: the bottom network receives static visual information at each time step and is trained to predict the next visual state; the top network receives input from the bottom network's hidden layer and is trained to generate the sequence of words which describes the current visual state. The network performed very well as it was able to generalize to patterns outside the training set. The syntax of the description sequences was also correct on every test and only a few semantic class errors were made.

Ghahramani & Allen (1991) designed a discrete-time learning algorithm, called Internal Target Generation (ITG), for an Elman ASRNN to solve the problem of sparse targets in temporal applications. The problem occurs due to ASRNNs not having adaptable recurrent connections that can correct errors in processing at one time step back. Thus by having cycles without any targets the input is left exposed to the noise from the feedback and the intervening inputs. The ITG algorithm generates targets during cycles wherein no targets are provided. For a two channel relative order problem the ITG algorithm performed consistently better than the Elman ASRNN using BP. For a temporal warping problem the BP ASRNN was slower at learning, but better at generalizing. They also developed a new ASRNN, called *Temporal Autoassociation* (TA) ASRNN (see Figure 2.3), which is designed to reproduce the current input and context units as additional outputs. This is achieved by adding a set of additional output units equal to the number of context and input units to an Elman ASRNN, thus forming an approximate inverse mapping from the hiddens to the new set of outputs. The TA ASRNN outperformed the Elman network for a temporal XOR and a delay line task.

Karjala *et al* (1992) trained an Elman ASRNN to rectify noisy inaccurate measurements for a simple dynamical system, a draining tank. The network was trained using the BFGS quasi-Newton nonlinear optimization algorithm to produce the reconciled simulated measurements, since this algorithm is much faster than BP (which uses gradient descent) for unconstrained optimization. Without any explicit knowledge of the nonlinear dynamics of the system, the Elman network was able to effectively reduce the noise level in the process measurements.

Fernando *et al* (1992) trained a two-hidden layer Elman ASRNN with feedback connections coming from the second hidden layer to detect high impedance faults in power distribution lines. The network was trained using discrete-time BP and although its performance was comparable with existing conventional algorithms, the former are capable of providing adaptive detectors

STUDY	ARCHITECTURE	LEARNING ALGORITHM
[Elman, 1988]	Elman network	Discrete-time BP
[Servan-Schreiber *et al*, 1988] [Cleeremans *et al*, 1989]	Elman network	Discrete-time BP
[Williams & Zipser, 1990a]	Elman network	Discrete-time hybrid FP-BPTT
[Cottrell & Tsung, 1991]	Elman network	Discrete-time BP
[Gordon *et al*, 1991]	Elman network	Discrete-time BP
[Iooss, 1991]	Elman network with self-looping context units	Discrete-time BP
[Ghahramani & Allen, 1991]	Elman network Temporal Autoassociation	Discrete-time ITG Discrete-time BP
[Bartell & Cottrell, 1991]	Two coupled Elman networks	Discrete-time BP
[Hoekstra & Kooijman, 1991]	Elman network	Discrete-time BP
[Debar & Dorizzi, 1992]	Elman having current and previous inputs	Discrete-time BP
[Karjala *et al*, 1992]	Elman network	BFGS quasi-Newton
[Fernando *et al*, 1992]	Two-hidden layer Elman network	Discrete-time BP
[Wang *et al*, 1993]	Two coupled Elman networks	Discrete time BP
[Psarrou & Buxton, 1994]	Elman network	Discrete-time BP
[Fanelli, 1994]	Elman network	Discrete-time BP
[McCann & Kalman, 1994]	Elman with input and contexts connected to outputs	Conjugate Gradient Learning
[Hester *et al*, 1994]	Combined Elman and RAAM network	Discrete-time BP

Table 2.2: Elman and related ASRNNs (studies in chronological order)

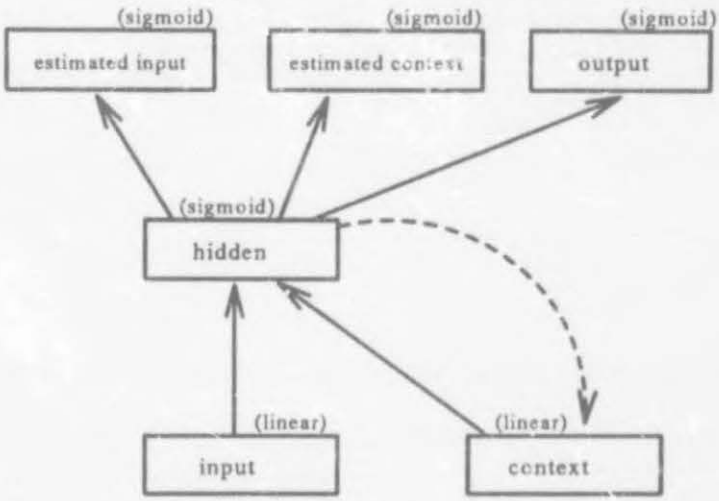


Figure 2.3: A Temporal Autoassociation ASRNN

which the latter lack.

Debar & Dorizzi (1992) used a modified version of the Elman ASRNN for time series intrusion detection. The network differs from the Elman ASRNN in that inputs are given at the current and previous time steps, the current inputs operate at the hidden level, and the current inputs have feedback connections to the previous inputs at the input level. The network performed very well when compared with classical algorithms and easily learned shifts in behaviour.

Wang *et al* (1993) constructed a modular ASRNN, which consists of two coupled Elman ASRNNs, for predicting successive elements of a Chinese word sequence. The bottom ASRNN learns to predict the major category of the next word, whereas the top ASRNN predicts the next possible word. The network achieved 85% correct prediction on the 50 training sentences used.

Psarrou & Buxton (1994) used the Elman ASRNN to address the problem of temporal prediction in computer vision applications and showed using discrete-time BP how observed trajectories can be mapped onto this network. Fanelli (1994) trained an Elman ASRNN on a simple bit string grammar and then studied the network's ability to simulate finite state automata in a stable manner while processing arbitrarily long input sequences. Stable regions for this network were identified using a subtractive algorithm, where each region corresponded to the states of the minimal finite automaton for the grammar.

McCann & Kalman (1994) presented a modified Elman ASRNN with extra connections from the input and context units to the outputs. The network was trained to differentiate between

English and French speakers, and provided for significant savings in time over the sequential version.

Hester *et al* (1994) proposed an ASRNN called *Predictive RAAM*, which is a combination of an Elman ASRNN and a RAAM (see section 1.2.2 for a brief description of RAAM networks), that can be used to discover and encode the most appropriate decomposition of the input sequences into words or characters. Using discrete-time BP the network obtained similar results compared to those of an Elman ASRNN for a word encoding application.

Other ASRNNs:

Other ASRNNs (see Table 2.3 for a summary) include those that do not have feedback connections that resemble the feedback of Jordan or Elman networks. Watrous & Shastri (1987) examined discrete-time BP networks with recurrent self-looping connections on sigmoidal hidden and output units for a speech recognition application. However, better results are obtained with Elman and Jordan related ASRNNs. Stornetta *et al* (1988) modified a standard three-layer BP network to include self-looping connections on the slow-decaying input units. This network provided good results when it was tested with applications such as motion detection, speech recognition, and signature verification. The work of Bachrach (1988) (self-recurrent output units) and Mozer (1988) (self recurrent hidden units) represent special cases of a general-purpose recurrent network using the discrete-time RTRL algorithm (see section 1.2.2).

Pollack (1990a) developed a second-order recurrent network, called the *Cascaded network*, using a discrete-time BP algorithm. This network has sigma-pi connections and linear state units that determine the weights of the network.

Leighton & Conrath (1991) proposed an *Autoregressive BP* network which is a recurrent network with each sigmoidal unit having k linear feedback connections from itself to it's own output value. Some of the properties of this approach were illustrated with trajectory prediction and sequence learning.

Winter (1991) described a trainable inference network (TIN) that learns state transition sequences of a sequential machine from examples of its behaviour. TIN is composed of two networks, TIN-1 and TIN-2, both using continuous-time ART learning. During training TIN-2 constructs state representations from machine behaviour examples, while TIN-1 produces sequences of transitions when given state representations. In an experiment TIN learned the behaviour of a machine that recognizes strings containing even numbers of both 1's and 0's.

STUDY	ARCHITECTURE	LEARNING ALGORITHM
[Watrous & Shastri, 1987]	Self-looping connections on hiddens and outputs	Discrete-time BP
[Stornetta *et al*, 1988]	Self-looping connections on slow-decaying inputs	Discrete-time BP
[Bachrach, 1988]	Output-to-output layer feedback	Discrete-time RTRL
[Mozer, 1988]	Hidden-to-hidden layer feedback	Discrete-time RTRL
[Pollack, 1990]	Cascaded network (sigma-pi connections)	Discrete-time BP
[Williams & Zipser, 1990a]	Output-to-output feedback	Discrete-time hybrid FP-BPTT
[Leighton & Conrath, 1991]	Autoregressive memory	DT Autoregressive BP
[Winter, 1991]	TIN network	Continuous-time ART
[Banzhaf, 1991a]	Coupled WTA, TIM, and WTA networks	Continuous-time Lateral Inhibition
[Hoekstra & Kooijman, 1991] [Hoekstra, 1992]	Adaptive general delayed links;	Discrete-time BP
[Diederich, 1992]	Stolcke network Weidi network	Discrete-time BP with fixed or adaptable learning rates
[Kobayashi & Hara, 1993]	Output-to-output feedback	Discrete-time BP
[Ryeu & Tak, 1993]	Output-to-Output feedback; no hidden layer	Discrete-time Perceptron

Table 2.3: Other ASRNNs (studies in chronological order)

Banzhaf (1991a) and Banzhaf & Kyuma (1991b, 1992) proposed a hybrid network consisting of a Winner-Take-All (WTA), Time-Intensity-Mapping (TIM), and another WTA network for the processing of spatio-temporal patterns. The first WTA network is used to recognize instantaneous spatial patterns and to select a winner unit via continuous-time lateral inhibition. The temporal sequence of patterns is mapped into intensities of the TIM units by coding the state of the WTA units by means of a simple growth/decay dynamics. The second WTA network is tuned to specific intensity patterns of TIM units, and by using lateral inhibition identifies the nearest of the known temporal patterns. This approach was tested with a problem of recognizing different transient sequences of letters. Feasible results and fine generalization were obtained.

Hoekstra & Kooijman (1991) described a general approach of recurrence with adaptable delayed links in multilayer networks for processing sequential data. Unlike the strict architectures of Jordan and Elman networks, recurrent connections in this approach are also allowed between layers and in all directions. Hoekstra & Kooijman defines a *delayed link* to be a connection between two units in which the output of the source unit at the previous time step is fed through a adaptable weight to the destination unit at the current time step. The weights of delayed links are also updated using the discrete-time BP algorithm. The approach is illustrated with sequential XOR, sequential encoder [Hoekstra & Kooijman, 1991], and counting [Hoekstra, 1992] problems. For the latter problem this mixed ASRNN outperformed the Elman network.

Diederich (1992) tested Jordan, Elman, Stolcke, and Weidi ASRNNs as tools for adaptive user interfaces. The Stolcke ASRNN has two clusters of linear state units, one on the level of the sigmoidal hidden units and the other on the level of the sigmoidal output units. Both the input units and hidden layer state units activate the hidden units, and the hidden units feed back to the state units. The hidden units together with the output layer state units activate the output units, and the output units feed back to activate their corresponding state units. All the feedback connections are fixed and one-to-one. The Weidi ASRNN has two hidden layers the first one linear, which feeds back one-to-one to a state layer, and the second one sigmoidal, which feeds forward to linear output units. The linear hidden units also feed forward one-to-one to the sigmoidal hidden layer. The one-to-one connections is also fixed. A discrete-time BP algorithm with fixed and adaptive learning rates were used. These two ASRNNs performed promising when they were compared with the Jordan and Elman ASRNNs for adaptive user interfaces, where the Weidi network provided the best results.

Kobayashi & Hara (1991) constructed an output-to-output feedback ASRNN that learns to recognize human emotions from facial expressions. The recurrent network was trained with

discrete-time BP and was found to perform almost similar recognition as those obtained by human beings. Ryeu & Tak (1993) proposed an output-to-output feedback ASRNN without a hidden layer to recognize Korean spoken digits. The network was trained with a Perceptron-type learning algorithm and achieved 92% accuracy on 600 training digits.

2.2 Discussion and Suggestions

It is evident from the overview of the ASRNN research literature that although the field is still in its infancy, ASRNNs have been used for a broad range of applications, a variety of different architectures and learning algorithms have been introduced, and they have shown very promising performance. According to Servan-Schreiber *et al* (1991), ASRNNs should be seen as a new entry into the taxonomy of computational machines (being more powerful in interesting ways than traditional finite state automata) and could prove sufficient for natural language processing.

The main advantages of architecture-specific recurrent neural networks include:

- computationally less expensive than general-purpose recurrent networks;
- less complex local interconnection structure than general-purpose networks;
- more stable than general-purpose recurrent networks;
- easier to study than other types of recurrent networks;
- although simpler, they retain much of the power of general-purpose recurrent networks;
- much more powerful than time-delay approaches, since they do not suffer from exponentially growing training sets and fixed time windows.

The following are some basic drawbacks of architecture-specific recurrent networks:

- restricted architecture;
- not general;
- compute error-gradient approximately;
- relative slow learning convergence and do sometimes not converge;
- and biologically unrealistic, if BP and its variants are employed.

In this study we are concerned with ASRNNs that use the discrete-time BP algorithm. The focus, in particular, is to explore the training, dynamics and complexity of the Elman, Jordan, and Temporal Autoassociation ASRNNs, since the first two networks are the most widely used ASRNNs and the latter is a promising extension of the Elman ASRNN. In addition, we suggest three alternative architectural designs for ASRNNs (see Figure 2.4): (a) An Output-to-Hidden Hidden-to-Hidden ASRNN; (b) an Output-to-Output Hidden-to-Hidden ASRNN; and (c) an Output-to-Hidden TA ASRNN. These three architectures combine each in an unique way the promising features of an Elman (the context layer), a Jordan (the state layer), and a TA ASRNN (the output units estimating the input and context values). The Output-to-Hidden Hidden-to-Hidden ASRNN is an expansion of the one-hidden-layer BP network to include a linear state layer with one-to-one recurrent connections from the sigmoidal outputs and a linear context layer with one-to-one recurrent connections from the sigmoidal hiddens. It is designed to incorporate both the internal representation and output history as input to the network, enabling it to learn temporal input and/or output sequences. For the Output-to-Output Hidden-to-Hidden ASRNN the linear state layer is not inserted at the input level but at the hidden level, so as to incorporate the internal network states as input and output history at the hidden level. The Output-to-Hidden TA ASRNN is a Temporal Autoassociation ASRNN to which a linear state layer is added that receives one-to-one recurrent connections from those sigmoidal output units not involved in learning the inverse approximation (not the output units involved in estimating the input and context values – see Figure 2.4). It is designed to give a TA network the additional benefit of being structured to deal with the production of output sequences. These ASRNNs along with the Elman, Jordan, and TA ASRNNs are compared in the next chapter using a variety of experiments. We also investigate the usefulness of output-to-output feedback connections by evaluating the training strategies for an Output-to-Output ASRNN.

The contributions of this study to the current ASRNN research literature, which is fairly experimental in nature, firstly start with the unique classification and overview given of the development and state of the art. In the next chapter a variety of ASRNNs are mutually compared, and one of our suggested architectures, the Output-to-Hidden Hidden-to-Hidden ASRNN, performed better than the Elman, Jordan, and TA network for a particular application. Regarding the classification dynamics analysis of ASRNNs, only Hierarchical Cluster Analysis (HCA) and Principal Component Analysis (PCA) were used in a few instances. We do not only use HCA extensively for a variety of applications, but also present Sammon Transformation Analysis [Sammon, 1969] as a superior clustering technique as opposed to PCA. We further determine the correspondence between the finite state machine that is simulated by the network and the one constructed for the training data. The learning dynamics of ASRNNs also did not receive

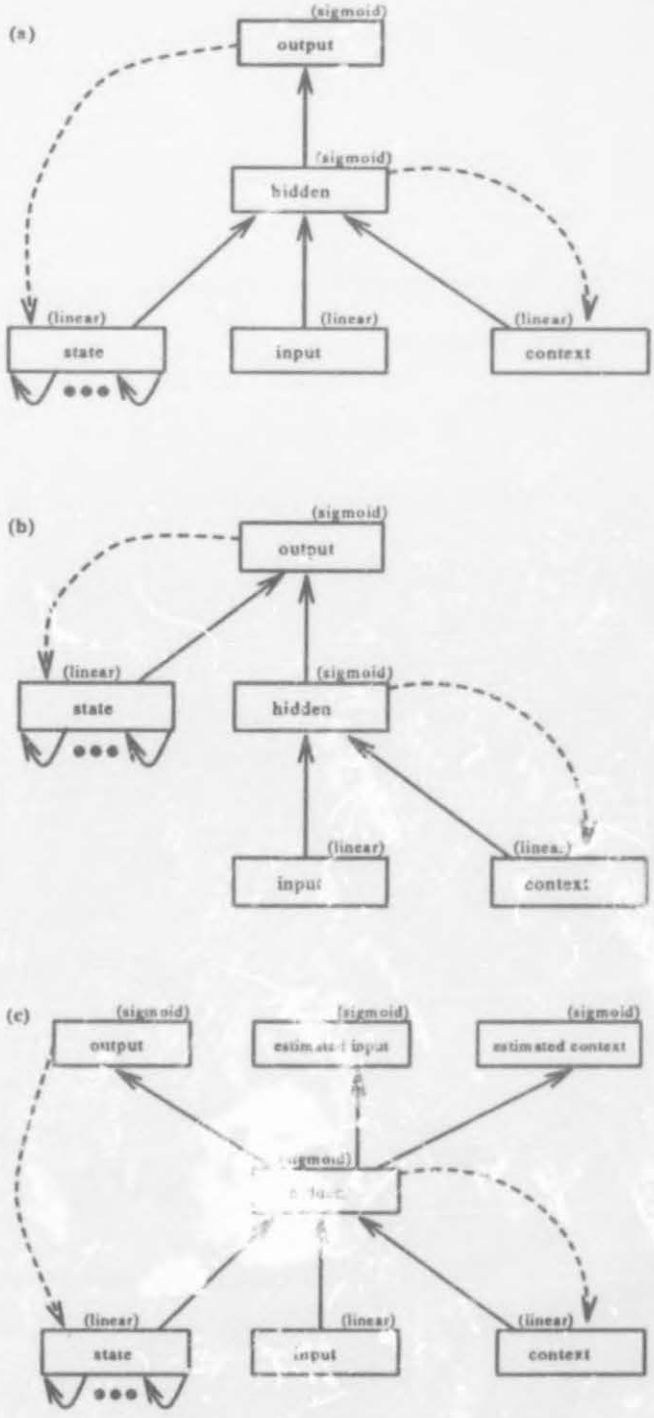


Figure 2.4: Suggested architectural designs for ASRNNs (a) An Output-to-Hidden Hidden-to-Hidden ASRNN; (b) An Output-to-Output Hidden-to-Hidden ASRNN; (c) An Output-to-Hidden TA ASRNN

much treatment. We respond by analyzing the evolution of clusters formed during learning and visualizing hidden activation time traces. Our complexity analysis of the capabilities, the number of hidden units, and capacity of ASRNNs is a pioneering effort to obtain theoretical results in this regard. Since training strategies thus far did not receive much attention in the neural network literature in general, those suggested in the next chapter would not only alleviate training difficulties and speed up learning for ASRNNs, but also for feedforward networks.

Chapter 3

Training Strategies

Much attention has been given to improving the learning time of neural networks trained with the popular back propagation algorithm. The reason is that because BP is a variation of Steepest Descent the inherent ill-condition of the algorithm makes it slow or prone to non-convergence. Previous investigations considered the initial values and changes to various parameters and variables of a neural network during training, such as the learning rate, momentum and weights [Silva & Almeida, 1990], to speed up learning. However, training strategies have mostly been overlooked as a possible method to address the problems of BP. For ASRNNs these deficiencies are more pronounced. The purpose of this chapter is firstly to develop training strategies for ASRNNs and feedforward networks that do not only alleviate these problems, but also lead to considerable improvement in training time. The second goal is to evaluate different ASRNNs with the training strategies by comparing the training performance of Elman, Jordan, and Temporal Autoassociation ASRNNs as well as other newly constructed ASRNNs (see section 2.2). Besides the problems with BP, the third goal is to address problems such as the effect of multiple attractor basins, the size of the initial subset, the increment in the subset size, the termination criteria for each subsequent subset, and the number of examples required for good generalization. The fourth goal is to examine the training strategies and ASRNNs with different applications, varying in complexity and ranging from sequence recognition to sequence generation. The final goal is to investigate the operation of the training strategies and determine why and how they lead to improved performance.

In this chapter we have developed *Increased Complexity Training*, two *incremental* training strategies and six *delta* training strategies, and showed that they improve training performance significantly and abate the BP problems for different ASRNNs as well as feedforward networks. We have also used the training strategies to compare different ASRNNs and found one of our sug-

gested architectures, the Output-to-Hidden Hidden-to-Hidden ASRNN, to be the most efficient one. The incremental training strategies also effectively address the problems mentioned in goal three. The training strategies and ASRNNs are evaluated with tasks involving only input sequences (Counting), input and output sequences (Addition), sequence generation (Digital Morse code generation), and sequence recognition (Temporal letter recognition). The feedforward networks are examined using a Cluster detection and Digit recognition task. The operation of the training strategies are investigated by visualizing the movement of hyperplanes and weights. The visualization graphs showed how the new training strategies efficiently reduce the weight activity and obtain faster convergence.

3.1 A Taxonomy of Training Strategies

For orientation purposes, a taxonomy of existing and newly suggested training strategies is presented. Examples of existing training strategies include Cottrell & Tsung (1991)'s *Combined Subset Training* (CST) and Elman (1991)'s strategy (discussed in section 3.2.11). CST first entails selecting randomly a manageable subset of training patterns to train the network initially. Then when the network has learned this set fairly well, select randomly another subset of equal size and add it to the first. Train the network further with the combined set. Repeat this procedure until the whole training set is included, or until the network is able to generalize to the rest of the original training set. In this chapter we introduce various training strategies and illustrate experimentally that with all other network parameters identical, the training strategy alone can speed up learning time considerably.

In the conventional *Fixed Set Training* (FST) the same set of training patterns are repeatedly presented during BP training until a success criterion is met. Our proposed training strategies investigate several points for variation of the training schedule: (1) construction of training subsets of the fixed set; (2) ordering of subsets according to a particular criterion; (3) termination criteria for each training subset; and (4) when weight updates occur. We briefly discuss these issues in turn, before elaborating on them in subsequent sections.

(1 & 2) Construction and ordering of subsets: There are many possible ways to construct and order subsets of the fixed training set. We have firstly proposed *Increased Complexity Training* (ICT) [Cloete & Ludik, 1993, Ludik & Cloete, 1993] in which the fixed set was partitioned into subsets and ranked, each subset being more complex than the previous. The complexity measure was problem-dependent, e.g. the number of output units switched on by patterns in a subset for

a counting problem, or subsets of one, two and three column examples for an addition problem. The network is first trained on the least complex subset. Then the second subset is randomly merged with the first, and the network trained again until successful. This process is repeated until the entire fixed set has been used. To circumvent the problem-dependent complexity measure, we have also proposed *Delta Subset Training* (DST) strategies [Cloete & Ludik, 1994b] in which examples are ordered according to inter-pattern distance, denoted by *delta*. We have investigated three different orders of example presentation using this schedule: *Smallest* difference between patterns (*Smallest Delta Subset Training* (SDST)), where patterns are ordered from the most difficult to the easiest to discriminate; *Largest* difference (*Largest Delta Subset Training* (LDST)), where patterns are ordered from the easiest to the most difficult to discriminate; and *Alternating* difference (*Alternating Delta Subset Training* (ADST)), where the first pattern has the smallest difference, second pattern the largest difference, third pattern the second smallest, etc.

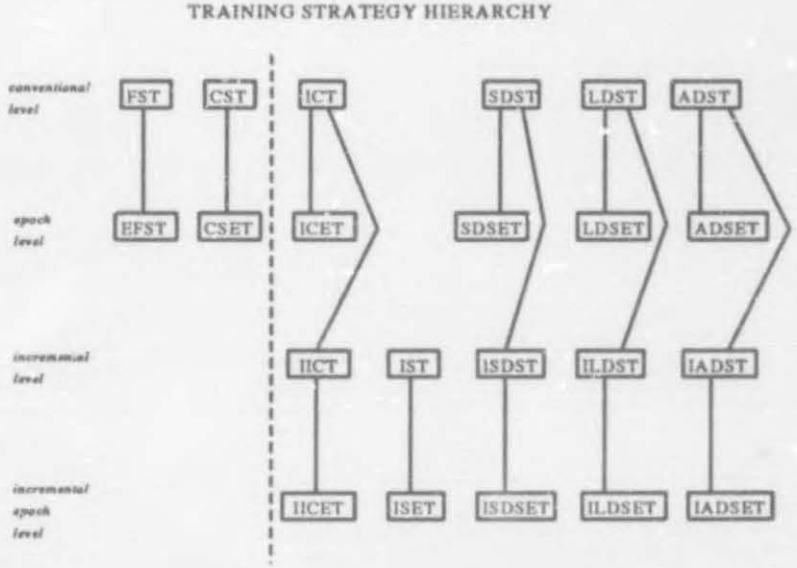


Figure 3.1: Development of Training Strategies

We further investigate *incremental* construction of subsets, whether complexity ordered or not. In the *incremental* training schedules, e.g. *Incremental Subset Training* (IST) [Cloete & Ludik, 1994a], the user specifies an increment (a number of examples) with which each subsequent training subset is enlarged by randomly merging the new training patterns. The initial subset contains the same number of patterns as the increment. Training occurs on the initial subset of training patterns until a success criterion is met, then on the next incrementally constructed subset, etc. For example, when incremental training is combined with ICT, denoted by *Incremental Increased Complexity Training* (IICT) [Ludik & Cloete, 1994], new

patterns to be merged are successively taken from the complexity ranked subsets in multiples of the increment, and similarly for DST strategies, which lead to incremental extensions of SDST, LDST, and ADST. For instance, ISDST denotes *Incremental Smallest Delta Subset Training.*

(3) Termination criteria: When training occurs on subsets of the fixed set, training each subset to the same strict precision as required for the fixed set, may cause the network to select a "local" solution instead of correctly generalizing to a global solution. To prevent this from happening we suggest that the initial error value obtained for the untrained network be linearly decreased, or decreased in proportion to the size of the subset and the fixed set, when calculating a required error value for termination of training on a particular subset. In this way the network is gradually restricted to find solutions of higher accuracy.

Furthermore, it is possible that a network can find a good solution without requiring training on the entire fixed set, i.e. a good subset of examples may exist within the fixed set. Therefore we added an outer training loop to the training schedule which terminates further training (and thus subset construction) when the success percentage on a representative test set reaches a user-specified value, thus indicating good generalization and simultaneously preventing overfitting of the data. Testing the network in feedforward mode is usually a quick operation which will not adversely affect the training time, and which has the advantage of detecting when a good set of training patterns has been found by terminating training at that point.

(4) Updating of weights: When using the conventional fixed set training, weights are usually updated after every single pattern, often referred to as "stochastic" weight update, or after presentation of all training patterns in the fixed set, known as updating after an "epoch" (also called "batch updating"). With the former type of weight update, the trained network may be skewed towards the most recent patterns in the epoch. Epoch updating counteracts this problem, although it may have an averaging effect on the training [Zurada, 1992]. Thus for each of the training strategies already discussed an "epoch update" variation was also investigated in which weights are only updated after presentation of the training subset. For instance, IICET denotes *Incremental Increased Complexity Epoch Training.*

In Figure 3.1 the hierarchy of training strategies are shown diagrammatically. New strategies are to the right of the vertical dotted line. The conventional level denotes weight updates after every single pattern, the epoch level refers to update after a subset, while the incremental level denotes the incremental construction of subsets.

3.2 Increased Complexity Training

The idea of *Increased Complexity Training* is simply that it should be quicker to learn the easy part of a problem first, and then gradually increase the complexity of the problem to be learned by giving "more difficult" training patterns in addition to those that the network have already learned. That means that the fixed training set be split into subsets of "increasing complexity", and having learned the initial subset, the next more complex subset is added to the first for subsequent training. This process is repeated until the whole training set is learned. So rather than selecting patterns randomly, patterns are selected "intelligently" by learning easier tasks first. This is reminiscent of human learning – for instance, first learn to count to small numbers, before proceeding to larger numbers.

In this section we demonstrate experimentally that training on increasingly more complex combined subsets reduces drastically the number of weight updates to reach a given Root Mean Square (RMS) error criterion, suggesting that easier subtasks should be learned first and the complexity of the task to be learned gradually increased. Training in this way much reduces the number of attractor basins (possible solutions) thus leading to faster convergence. Convergence time is related to the complexity of the problem, therefore learning easier subproblems first constrains the solution by eliminating many possibilities, causing faster convergence. For ICT and CST, we investigate various schedules of RMS error values to serve as termination criteria for each subsequent subset. We compare these two training strategies with FST for six different ASRNNs and a feedforward network using three applications. For ASRNNs we distinguish between single patterns, presented at each time step, and temporal patterns consisting of an entire sequence of single patterns which has to be learned. For all comparisons the initial conditions and parameters were identical for each network, i.e. the same initial weights and the same learning and momentum rates are used. In sections 3.2.1 to 3.2.6 different ASRNNs are evaluated with the Counting application and then compared in section 3.2.7 for each training strategy.

3.2.1 Elman ASRNN : COUNTING

For the *counting* application [Hoekstra, 1992] an ASRNN sequentially learns to count pulses from zero to 15. The absence of a pulse is represented by a zero, and resets the counter. The network receives one input at a time, and outputs the binary representation of the counter, i.e. four bits. The counting task and Elman ASRNN, with one input unit, eight hidden and eight

context units, and four output units, are illustrated in Figure 3.2.

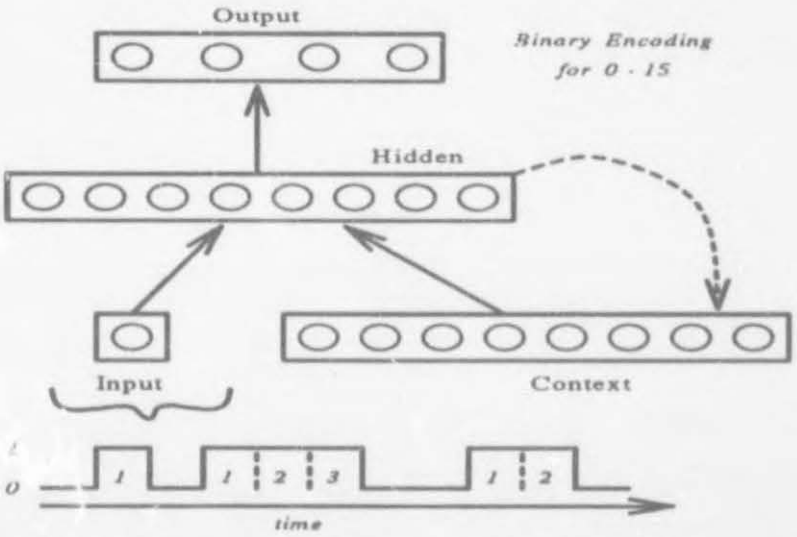


Figure 3.2: Elman ASRNN for the Counting application

For ICT four training sets, numbered one to four of increasing complexity, were generated as follows. The first set contains two sequential patterns learning to count from 0 to 1, i.e. four single patterns, which requires only the first output unit to vary; the remaining three output a 0. The next training set includes the first in random order and contains sequential patterns to count from 0 to 3, i.e. vary both the first and second output units, and so on for the next two training sets as illustrated in Figure 3.2. The fixed set contains all 16 temporal patterns (i.e. 137 single patterns) in random order. Four random combined subsets containing 2, 4, 8 and 16 temporal patterns each were generated for experimentation using *CST*.

For our initial experiments, training was continued for each subset until either a 100% success ratio on the training set was reached, or the training performance did not improve. Weights were updated after every input and the optimal learning and momentum rate was determined on the fixed training set and then used for all experiments. Although ICT achieved a 25% improvement compared to FST, it appears from Table B.1 (which contains a summary of simulation results – see Appendix B) that there is a decrease in the number of epochs for ICT from the first to the second more complex subset. This indicates that the network is erroneously trying to adapt the weights to force the higher-order 3 digits of output to zero – a result that would have to be "unlearned" when more complex training sets are used. The answer is therefore to terminate training earlier by requiring a lower success ratio on the training set, and gradually increase this termination criterion. Thus the experiment was repeated and training was terminated where a success ratio of greater than or equal to 70%, 85%, 85% and 100% for the respective subsets

were obtained (the results are given in Table B.2). In this case ICT improved training time by 50% (35 epochs vs. 71) compared to training on a fixed set, and by 34% (35 epochs vs. 53) over its previous value when an *incremental success ratio* (*isr*) is used. For CST comparison with its previous result (not using *isr*) shows a 26% improvement, but again the total number of epochs required is much more than that using ICT. See [Cloete & Ludik, 1993] for a more detailed discussion on these experiments.

We then pursued a more effective way of the determining the termination criterion for each subsequent subset by investigating various schedules of Root Mean Square error values for ICT and CST. Since the size of the training sets vary, the number of epochs would be an unfair measure for convergence time. The training progress was therefore measured in terms of number of weight updates (i.e. single pattern presentations) required to reach the required RMS termination value. The optimum RMS termination values obtained for ICT and CST respectively are $0.55 - 0.45 - 0.35 - 0.15$ and $0.65 - 0.55 - 0.45 - 0.15$, where a dash separates the RMS termination values of each subsecuent training subset. These RMS values were obtained out of combinations tried from 0.40 to 0.70 for the first subset, 0.35 to 0.60 for the second, and 0.25 to 0.45 for the third. In every schedule the RMS termination criterion for a particular subset influences the convergence of subsequent training subsets.

For each training strategy, ten simulations were performed with the same initial conditions, except for different initial sets of weights. The average and standard deviation (Std. D.) of the number of updates, as well as the average and standard deviation of the sum of the Euclidean distance of all the weight changes (indicated by $\sum E_A$) were determined. The summary of the simulation results appears in Table 3.1.

Training Strategies	Average # updates	Std. D. # updates	Average $\sum E_A$	Std. D. $\sum E_A$	Best # updates	% improvement compare to FST
ICT	17620	6978	568.37	184.05	9465	44.3%
CST	18260	12116	683.53	590.15	7878	42.2%
FST	31606	30182	945.58	734.17	8494	-

Table 3.1: A comparison of the average *counting* simulation results for the Elman ASRNN

From Table 3.1 it is evident that ICT and CST outperforms FST on average by respectively 44.3% and 42.2%, with ICT being the most consistent by having the lowest standard deviation. The worst case results for FST, CST and ICT were respectively 90968, 53174, and 36487, explaining the large standard deviation for FST. ICT also achieved the lowest total sum of

Euclidean weight changes and the lowest number of updates on average. The graph in Figure 3.3 highlights the results of a particular simulation for the counting, where ICT improves the number of updates by 13% compared to CST, and by 8% for the FST. The jumps on the error curves in Figure 3.3 roughly correspond to the incremental modification of the training subsets, i.e. the addition of new temporal patterns.

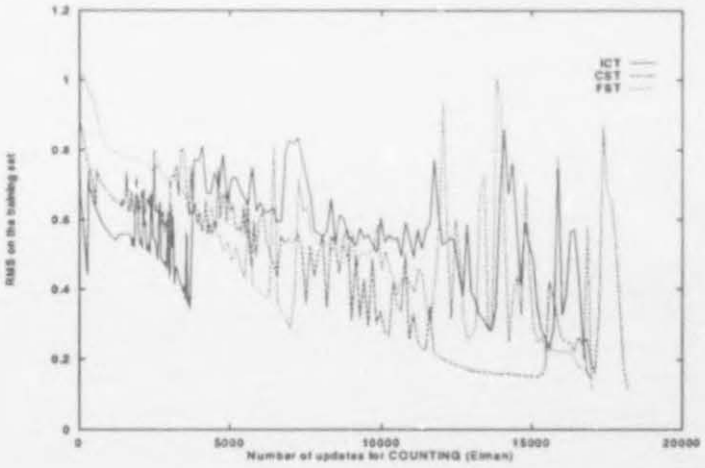


Figure 3.3: The performance of the Elman ASRNN for Counting

3.2.2 Temporal Autoassociation ASRNN : COUNTING

The second ASRNN for which the training strategies was evaluated, is the Temporal Autoassociation ASRNN [Ghahramani & Allen, 1991] (presented in section 2.1), which is designed to reproduce the current input and context units as additional outputs. The same counting experiment was repeated for the TA ASRNN with identical training sets and parameters used for the Elman ASRNN. The TA network consisted of one input unit, eight hidden and eight context units, and 13 output units (4 for representing the binary sum of inputs and the rest for reproducing the inverse mapping).

The optimum RMS termination values obtained for ICT and CST were respectively, 0.50−0.40−0.30−0.15 and 0.65−0.55−0.45−0.15. The RMS criterion for each subset for ICT was more strict in every case compared to those for the corresponding CST subsets. This is attributed to the more complex earlier subsets of CST where sets are not increasing in complexity, but random. For each training strategy, ten simulations were performed with the same initial conditions, except for different initial sets of weights. Again the average and standard deviation of the number of updates, as well as the average and standard deviation of the sum of the Euclidean

distance of all the weight changes were determined.

Training Strategies	Average # updates	Std. D. # updates	Average $\sum E_A$	Std. D. $\sum E_A$	Best # updates	% improvement compare to FST
ICT	7778	1241	349.68	52.22	5616	46.9%
CST	12131	3322	511.45	108.18	7756	17.2%
FST	14645	16153	436.91	123.87	5206	-

Table 3.2: A comparison of the average *counting* simulation results for the Temporal Autoassociation ASRNN

The simulation results (indicated in Table 3.2) show an impressive performance for ICT. Although CST improved performance by 17.2% compared to FST, its total sum of Euclidean weight changes are higher than FST. This is attributed to the incremental addition of new temporal patterns which roughly corresponds with the large jumps on the RMS error curve illustrated in Figure 3.4. The graph in this figure also highlights the results of a particular simulation, where ICT improved performance by a remarkable 85% compared to FST and 15% to CST.

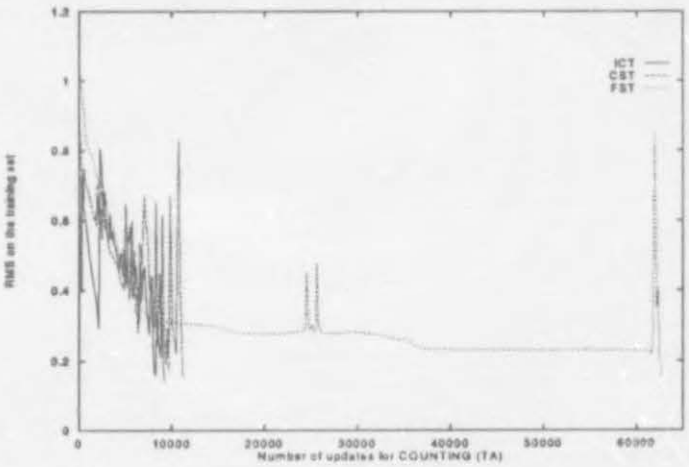


Figure 3.4: The performance of TA for Counting

In spite of the larger network for TA as well as the additional task of learning an inverse mapping of the current input and context units, ICT improved performance by 56% (7778 versus 17620) compared to ICT for the Elman ASRNN. We elaborate on this in section 3.2.7, where we compare more thoroughly the performances of the different ASRNNs for counting.

3.2.3 Jordan ASRNN : COUNTING

ICT, CST and FST were also evaluated for the Jordan ASRNN [Jordan, 1986] (presented in section 2.1) with identical training sets and parameters used for the counting task as in the Elman and TA ASRNNs. The Jordan network consisted of one input unit, eight hidden units, four output units, and accordingly four state units.

The RMS termination for the final subset was again 0.15, with the optimum RMS termination values obtained for the first three subsets of ICT and CST respectively, 0.70 – 0.65 – 0.60 and 0.70 – 0.60 – 0.50. The RMS criteria for ICT and CST were less strict compared to those for the Elman and TA ASRNNs, indicating that the Jordan ASRNN found the counting task more difficult to learn. The graph in Figure B.1 illustrates the results of a simulation where ICT improved performance by 21.3% (35675 versus 45347) compared to FST and 12.4% (35675 versus 40720) to CST. The learning curve for each strategy emphasizes the network's difficulty with the task, where they initially start smoothly before lowering the RMS value in a fairly "chaotic" manner. The average simulation results is summarized in Table 3.3.

Training Strategies	Average # updates	Std. D. # updates	Average $\sum E_A$	Std. D. $\sum E_A$	Best # updates	% improvement compared to FST
ICT	72391	35853	2618.98	1411.10	34462	20.7%
CST	79858	35975	2818.54	1141.37	28596	12.5%
FST	91283	38568	2965.41	948.34	45347	-

Table 3.3: A comparison of the average *counting* simulation results for the Jordan ASRNN

The much larger average number of updates obtained for the Jordan ASRNN compared to the Elman and TA ASRNN is attributed to the former having only four state units as opposed to the eight context units of the latter (implying a less powerful memory) as well as having only an output history at its disposal. ICT showed that although the Jordan ASRNN is not suited to perform well for Counting (this application involves learning input sequences and not output sequences), it still leads to improved performance when compared to the other training strategies.

3.2.4 Output-to-Hidden Hidden-to-Hidden ASRNN : COUNTING

The training strategies were also evaluated with the Output-to-Hidden Hidden-to-Hidden ASRNN (one of the new ASRNNs proposed in section 2.2), which is designed to incorporate both internal representation and output history as input to the network. The counting experiment was repeated for this ASRNN with identical training sets and parameters used for the previous networks. The ASRNN consisted of one input unit, eight hidden and eight context units, and four output and four state units, with the context and state units operating on the same level as the input.

The optimum RMS termination values obtained for ICT and CST were respectively, 0.50 – 0.45 – 0.40 – 0.15 and 0.65 – 0.55 – 0.45 – 0.15. Again the RMS criterion for each subset for ICT was more strict in every case compared to those for the corresponding CST subsets. The summary of the average simulation results appears in Table 3.4.

Training Strategies	Average # updates	Std. D. # updates	Average $\sum E_A$	Std. D. $\sum E_A$	Best # updates	% improvement compared to FST
ICT	5593	1190	244.55	68.14	4012	12.9%
CST	6069	1992	262.68	87.05	4427	5.5%
FST	6425	1541	271.33	66.88	4110	-

Table 3.4: A comparison of the average *counting* simulation results for the Output-to-Hidden Hidden-to-Hidden ASRNN

ICT needed the lowest total sum of Euclidean weight changes to achieve the lowest number of updates on average. It also attained the best number of updates for a particular simulation (4012), although the other strategies also faired well in this respect. The graph in Figure 3.5 highlights the results of a particular simulation, where ICT improves the number of updates by 51.2% (4012 versus 8220) compared to FST, and by 25.1% (4012 versus 5360) to the CST. Note the three clear jumps on the error curve of ICT, which correspond to the three subsets of increasing complexity being introduced to the network. The very good simulation results obtained with all the training strategies show that the Output-to-Hidden Hidden-to-Hidden network is indeed a very promising ASRNN. This performance is attributed to the network's ability to deal effectively with input sequences by having a context layer on the input level (similar to the Elman ASRNN).

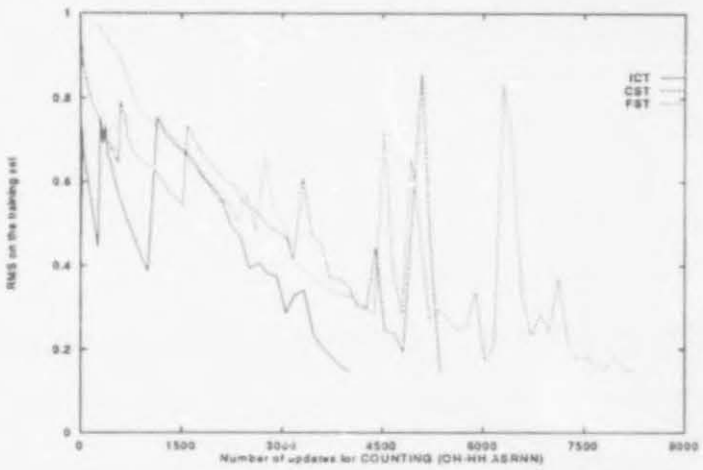


Figure 3.5: The performance of the Output-to-Hidden Hidden-to-Hidden ASRNN for Counting

3.2.5 Output-to-Output Hidden-to-Hidden ASRNN : COUNTING

Another newly proposed ASRNN, the Output-to-Output Hidden-to-Hidden ASRNN (section 2.2), designed to incorporate internal representation history at the input level and output history at the hidden level, was also employed to evaluate the training strategies. The counting experiment was repeated for this ASRNN with identical training sets and parameters used for the previous networks. The ASRNN consisted of one input unit, eight hidden units, eight context units at the input level, four output units, and four state units operating at the hidden level.

From an ASRNN point of view, it was also interesting to determine what the effect of inserting output history at the hidden level would have on the network performance for each training strategy. From the simulation results it were evident that the network found the counting task very difficult to learn and seldom reached the desired RMS value for 0.15. After using a RMS termination value of 0.35 for the final subset, we have obtained optimum RMS termination values of $0.55 - 0.50 - 0.45$ and $0.70 - 0.60 - 0.45$ for the first three subsets of ICT and CST respectively. Again the RMS criteria for ICT were more strict than those for CST. The average simulation results are summarized in Table 3.5.

It follows from Table 3.5 that ICT and CST outperforms FST on average, where ICT achieved the best number of updates (8077) for a particular simulation The graph in Figure B.2 highlights the results of a simulation, where ICT improves the number of updates by 35.2% compared to FST, and by 19.5% to the CST. The relatively bad performance of this network compared to the other ASRNNs is attributed to the former having state units operating at the hidden level,

Training Strategies	Average # updates	Std. D. # updates	Average $\sum E_A$	Std. D. $\sum E_A$	Best # updates	% improvement compared to FST
ICT	27034	13220	996.22	453.61	8077	26.1%
CST	29812	12382	1255.47	433.62	11340	18.5%
FST	36579	40125	1556.40	1534.48	11645	-

Table 3.5: A comparison of the average *counting* simulation results for the Output-to-Output Hidden-to-Hidden ASRNN

which corrupt the internal representation being fed forward to the outputs, since the output history is not useful in remembering the previous temporal input of counting. We conjecture that all ASRNNs with this type of connection would fair poorly for tasks having a temporal input such as counting.

3.2.6 Output-to-Output ASRNN : COUNTING

The training strategies were also evaluated with the Output-to-Output ASRNN (section 2.2), which is designed to incorporate only output history at the hidden level. We also test our conjecture that ASRNNs with output-to-output connections would fair badly for tasks having temporal input as opposed to temporal output. The counting experiment was repeated for this ASRNN with identical training sets and parameters used for the previous networks. The ASRNN consisted of one input unit, eight hidden units, four output units, and four state units operating at the hidden level.

The simulation results in Figure 3.6 showed that this ASRNN was unable to learn the counting task, thereby confirming our conjecture that the ASRNN would perform poorly. From the figure it is interesting to note that with ICT and CST, the ASRNN initially tried to learn the earlier subsets. For example, for ICT, the RMS value was decreased for the first subset from 0.89 to 0.47, for the second subset from 0.71 to 0.49, for the third from 0.79 to 0.47, and the fourth from 0.82 to 0.71. Although output-to-output connections did not prove to be useful for tasks with temporal input, it should still be useful for tasks having a fixed input and temporal output.

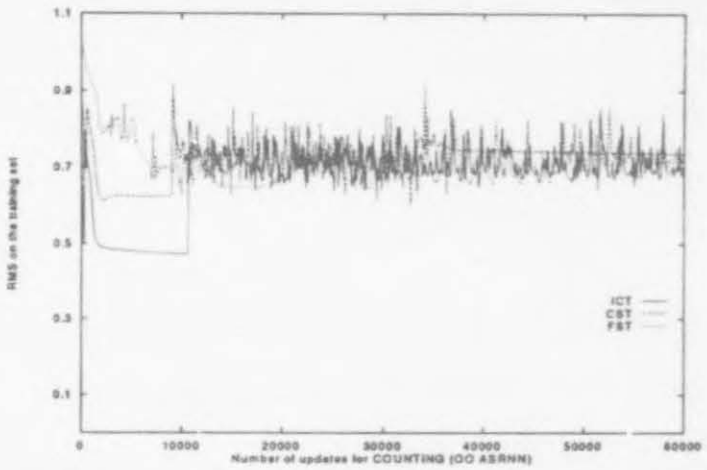


Figure 3.6: The performance of the Output-to-Output ASRNN for Counting

3.2.7 Comparison of different ASRNNs for COUNTING

In this section we compare the performance of different ASRNNs for the counting task. The training strategies are compared in terms of the number of weight updates (i.e. single pattern presentations). However, this measure would be an unfair measure to compare different ASRNNs, since the number of weights per network varies. For a more fair comparison we have to multiply the average number of updates by the number of weights in each network to obtain the total number of weights updated. This measure is related to the actual convergence time and the number of computations performed by each network. For each network the optimal network size was experimentally determined. Therefore the total number of weights updated represent the best performance for each particular network. Since the desired RMS termination value for the final subset of the Output-to-Output Hidden-to-Hidden ASRNN was not the same as the one used for the other ASRNNs and the simulations for the Output-to-Output ASRNN did not converge, we only compare the Elman, Temporal Autoassociation, Jordan, and Output-to-Hidden Hidden-to-Hidden ASRNNs for each training strategy. The comparison results appear in Table 3.6.

It follows from Table 3.6 that the Output-to-Hidden Hidden-to-Hidden (O-H H-H) ASRNN outperforms the Jordan, Elman, and TA ASRNNs for FST by respectively 86.7%, 73.4% , and 66.1%. The respective percentages for O-H H-H's ICT are 85.4%, 58.5%, and 44.4%, and for CST are 85.6%, 56.5% and 61.3%. Compared to the Jordan ASRNN, the O-H H-H ASRNN improved the number of weights updated for all three training strategies consistently by more than 85%. Although the TA ASRNN improved the performance of FST and ICT when compared to the

ASRNNs	Number of network weights	FST		CST		ICT	
		# weights updated	% improve (Jordan)	# weights updated	% improve (Jordan)	# weights updated	% improve (Jordan)
O-H H-H	136	873800	86.7%	825384	85.6%	760648	85.4%
TA	176	2577520	60.8%	2135056	62.9%	1368928	73.7%
Elman	104	3287024	50.0%	1899040	67.0%	1832480	64.8%
Jordan	72	6572376	–	5749776	–	5212152	–

Table 3.6: A comparison of the different ASRNNs for counting

Elman ASRNN, the opposite is true for CST. From this counting results we assert that the newly proposed Output-to-Hidden Hidden-to-Hidden ASRNN indeed distinguished itself as the best ASRNN. Since the O-H H-H ASRNN incorporates the history of the internal representations as well as the output, causing this network to be able to learn temporal input and/or output sequences, we conjecture that this general-purpose ASRNN would be able to learn most temporal tasks of this kind very efficiently.

In sections 3.2.8 to 3.2.10 the most widely used ASRNNs, Elman, Jordan, and TA ASRNNs are evaluated with the more complicated Addition application (input and output sequences) for each training strategy.

3.2.8 Elman ASRNN : ADDITION

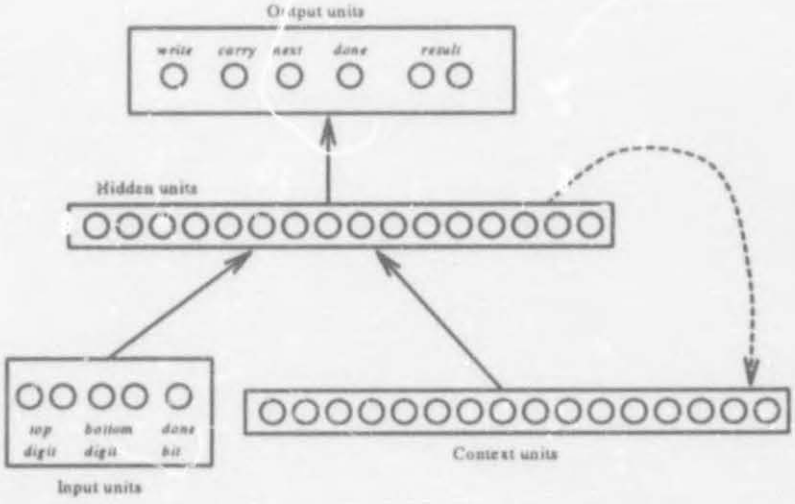


Figure 3.7: Elman ASRNN for the Addition application

For the *addition* experiment [Cottrell & Tsung, 1991] the aim is to learn to sequentially add two base four numbers. Each base four number is given a two-digit binary representation. The Elman ASRNN, illustrated in Figure 3.7, is confined to one column of digits at a time. It has five inputs, four representing the one column of digits and one indicating the end of the input, 16 hidden and 16 context units, and six output units representing the sum (two units) of the one column of digits and the four possible actions (four units). Actions are to write the sum, to remember or output the carry, shift to the next column of digits, and indicate if done. The addition task is very complex, because not only must it learn a sequence of inputs, but for each input a different sequence of outputs is required. See Figure B.3 for the program code and an example of the input and target output sequences involved in a three column addition.

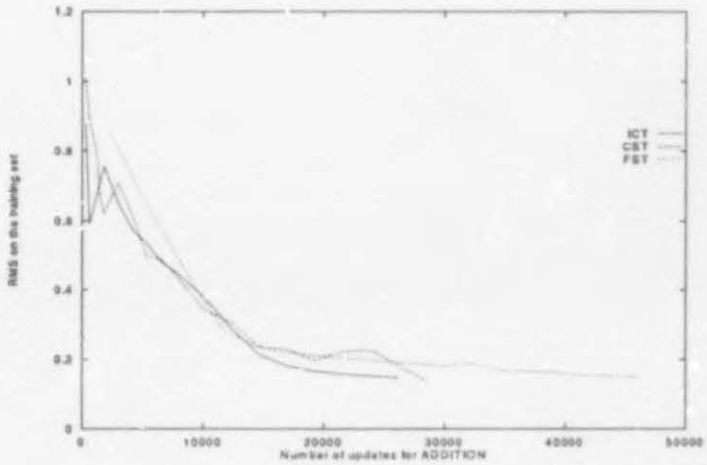


Figure 3.8: The performance of the Elman ASRNN for Addition

For one-, two- and three-column addition there are 4096 different temporal patterns. The training set for FST consisted of 8% of this set, constituting 320 temporal patterns, whereas the independent test set consisted of the other 92% of the total set. For ICT three training subsets of increasing complexity were generated, corresponding to 1%, 4% and 8% of the total training set, and each set containing patterns for adding, respectively, one column of figures, two and three columns. For CST the three subsets contained the same number of temporal patterns as each ICT subset, but randomly chosen. Again all network initial conditions for each experiment were identical. For comparison purposes, the final subsets of ICT and CST were the same one used for FST.

The following two termination criteria were used: A RMS error value of ≤ 0.15 for the final subset, and a success percentage on the independent test set of $\geq 97\%$. The optimum RMS termination values obtained for ICT and CST were both $0.65 - 0.45 - 0.15$. Ten simulations

were performed for each training strategy with the same initial conditions, except for different initial weights. For a particular simulation, illustrated in Figure 3.8, ICT improved the number of updates by 46% compared to FST and by 8% compared to CST.

Training Strategies	Average # updates	Std. D. # updates	Average $\sum E_A$	Std. D. $\sum E_A$	Best # updates	% improvement compared to FST
ICT	36186	9388	1324.15	253.46	22735	21.1%
CST	39478	20579	1367.41	279.77	25726	13.9%
FST	45870	16612	1540.02	380.65	29965	-

Table 3.7: A comparison of the average *addition* simulation results for the Elman ASRNN

From Table 3.7 it is apparent that ICT outperforms FST and CST on average by respectively 21.1% and 8.3%, with ICT accordingly being the most consistent by having the lowest standard deviation (9388 updates). The best number of updates for each strategy is proportionate to the average number of updates obtained for each one of them, with ICT achieving the best result of 22735 updates. Both ICT and CST reduce the large number of different attractor basins due to multiple solutions, thus lessening the "wandering about" in the solution space discussed by Kolen & Pollack (1991). Convergence time is related to the complexity of the problem [Judd, 1988], therefore learning easier subproblems first constrains the solution space by eliminating many possibilities, causing faster convergence. The elimination of many possibilities also reduces unnecessary weight changes. This can be visualized in Figure 3.9, where the weight changes of a particular weight during training (same simulation as in Figure 3.8) are pictured for FST, ICT and CST. Note how quickly ICT's weight changes stabilize as opposed to that of FST and CST. The Euclidean distance of weight changes per update for each strategy, illustrated in Figure B.5, further emphasizes the reduction in weight activity for ICT and CST.

3.2.9 Temporal Autoassociation ASRNN : ADDITION

The training strategies were also evaluated with the Temporal Autoassociation ASRNN for addition. The TA network consisted of five input units, 16 hidden and 16 context units, and 27 output units (six units representing the sum and actions, and the rest for reproducing the inverse mapping). The addition experiment was repeated for this ASRNN with identical training sets and parameters used for the Elman ASRNN. The simulation results in Figure B.4 showed that the addition experiment for the TA ASRNN never converged, no matter what training strategy was used. From the figure we note that with ICT, the ASRNN initially tried to learn the earlier

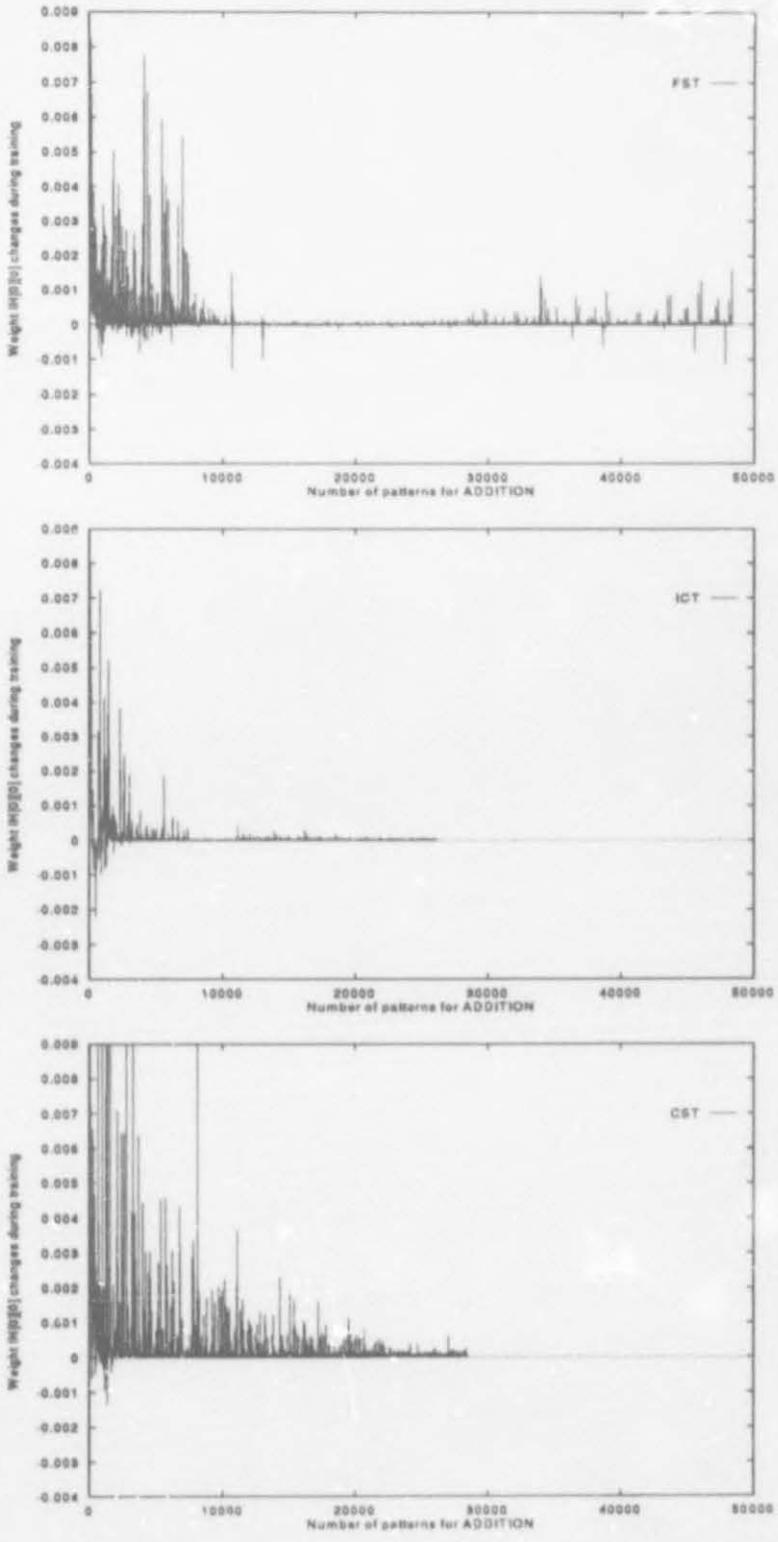


Figure 3.9: Visualization of the weight changes of a particular weight during FST, ICT and CST training for Addition (Elman ASRNN)

subset. For example, for ICT, the RMS value was initially decreased from 0.91 to 0.58, before it stabilized around 0.85. The TA ASRNN is an unsuitable architecture for the addition problem, since it constrains the solution space to those solutions for which a kind of inverse mapping exists. For addition many different numbers sum to the same result, thus being a many to one mapping.

3.2.10 Jordan ASRNN : ADDITION

ICT, CST and FST were also evaluated for the Jordan ASRNN with identical training sets and parameters used for the addition task in the Elman and TA ASRNNs. The Jordan network consisted of five input units, 16 hidden units, six output and six state units. The RMS termination for the final subset was again 0.15, with the optimum RMS termination values obtained for the first two subsets of ICT and CST respectively, $0.55 - 0.45$ and $0.65 - 0.45$. The average simulation results are summarized in Table 3.8.

Training Strategies	Average # updates	Std. D. # updates	Average $\sum E_A$	Std. D. $\sum E_A$	Best # updates	% improvement compared to FST
ICT	122698	29131	2431.13	402.71	67591	9.9%
CST	135242	17248	2697.54	234.50	113552	0.7%
FST	136226	18678	2785.21	351.84	103725	-

Table 3.8: A comparison of the average *addition* simulation results for the Jordan ASRNN

It follows from Table 3.8 that ICT outperforms FST and CST on average by respectively 9.9% and 9.3%, but also proved to be less consistent by having the highest standard deviation. The reason for the latter is partly due to ICT's best number of updates being a relative low value of 67591. Again the Jordan ASRNN, having only six state units as opposed to 16, obtained a much larger average number of updates compared to the Elman ASRNN. However, to put these results in a fair perspective, we have to apply the same technique as in section 3.2.7 (i.e. multiply the average number of updates by the number of weights in each network to obtain the total number of weights updated). We then obtain improvements for the Elman ASRNN of 46.5%, 53.6%, and 53.2% for respectively FST, CST, and ICT, when compared to the Jordan ASRNN. This again indicates the Elman ASRNN's superiority over the Jordan ASRNN for tasks having temporal input sequences. The graph in Figure B.5 illustrates the results of a simulation where ICT improved performance by 36.9% (103231 versus 163655) compared to FST and 24.8% (103231 versus 137200) compared to CST. The exponential nature of the learning curve for each

strategy emphasizes the network's relative quick mastering of the initial easier subsets before experiencing much difficulty with the larger (in ICT's case also more complex) subsets.

To illustrate that Increased Complexity Training is not only effective for ASRNNs, the next section evaluates the training strategies for a feedforward BP network.

3.2.11 Feedforward BP : Cluster Detection

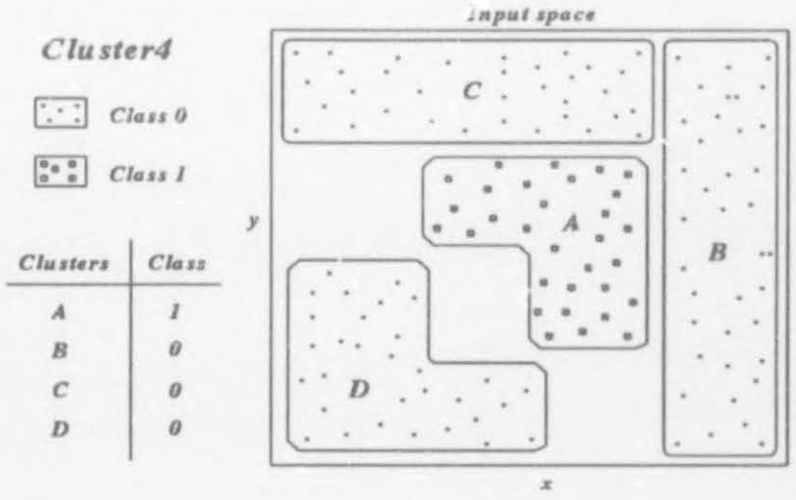


Figure 3.10: The Cluster Detection experiment: Detecting two classes in four clusters of input

The training strategies are evaluated here for a feedforward BP network using the Cluster detection experiment, where the aim is to detect two classes in four clusters of input, as illustrated in Figure 3.10. The feedforward BP network consisted of two inputs, representing the x and y coordinates of a point in the input space, three hidden units, and two output units, representing a class of type zero or one. A total set of 1300 points were generated from the four clusters in the input space. The training set for FST consisted of 20% of this set, constituting 260 input patterns (130 patterns from each class), whereas the independent test set consisted of the other 80% of the total set (1040 single patterns). For ICT three training subsets of increasing complexity were generated, corresponding to clusters $A\&B$ (94 single patterns), clusters $A\&B\&C$ (176 single patterns) and clusters $A\&B\&C\&D$ (260 single patterns). For CST the three subsets contained the same number of single patterns as each ICT subset, but randomly chosen from the four clusters. All network initial conditions for each experiment were identical. For comparison purposes, the final subsets of ICT and CST were the same one that were used for FST.

The following two termination criteria were used: A RMS error value ≤ 0.20 for the final subset,

Training Strategies	Average # updates	Std. D. # updates	Average $\sum E_A$	Std. D. $\sum E_A$	Best # updates	% improvement compared to FST
ICT	30199	3945	712.81	76.98	25314	46.6%
CST	34395	4206	798.36	130.04	26734	39.2%
FST	56550	19339	1214.68	390.49	31200	-

Table 3.9: A comparison of the average Cluster Detection simulation results for the Feedforward BP network

and a success percentage on independent test set of $\geq$ 90%. Since epoch updating seems to perform very well for feedforward networks in practice, we have also evaluated ICET, CSET, and FSET. We note that epoch updating does not prove to be that effective for ASRNNs in practice [Diederich, 1992], explaining why we did not evaluate them for ASRNNs. The optimum RMS termination values obtained for ICT (also ICET) and CST (also CSET) were respectively 0.65 – 0.45 – 0.20 and 0.70 – 0.50 – 0.20. The simulation results of ICT, CST and FST are summarized in Table 3.9.

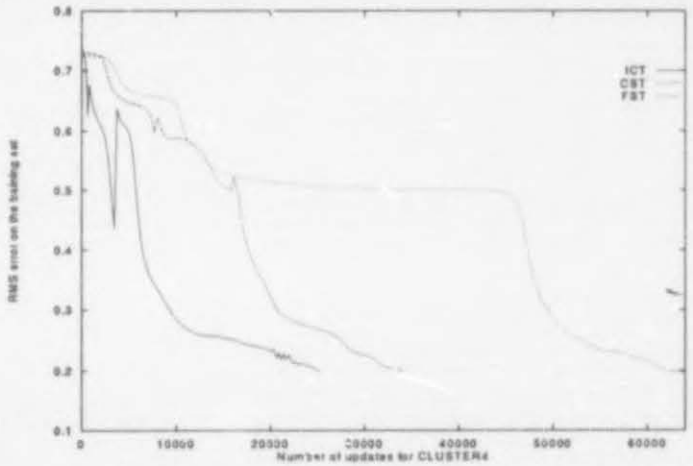


Figure 3.11: The performance of the Feedforward BP network for Cluster Detection

From Table 3.9 it is apparent that ICT outperforms FST and CST on average and also attained the best number of updates for a particular simulation (25314). For a particular simulation, illustrated in Figure 3.11, ICT improved the number of updates by 60.3% (25314 versus 63700) compared to FST and by 28.8% (25314 versus 35555) compared to CST. Note the three clear jumps on the error curve of ICT, which correspond to the three subsets of increasing complexity being introduced to the network.

Training Strategies	Average # patterns	Std. D. # patterns	Average $\sum E_A$	Std. D. $\sum E_A$	Best # updates	% improvement compared to FST
ICET	22035	10783	63.23	11.59	14526	61.0%
CSET	36909	17606	64.35	9.69	20296	34.7%
FSET	38740	7920	72.40	7.60	29900	31.5%

Table 3.10: A comparison of the average Cluster Detection simulation results for the Feedforward BP network – epoch updating

The simulation results for ICET, CSET, and FSET appear in Table 3.10, where the improvement percentages showed in this table indicate each training strategy's improvement compared to FST. From this table it is evident that better results are obtained with epoch updating as opposed to updating after every pattern. ICET improved performance compared to ICT by 27.0%, and FSET compared to FST by 31.5%. However, CSET's performance was 6.8% worse than CST. Also note that the averages and standard deviation of the sum of the Euclidean distance of all the weight changes is much lower than those in Table 3.9. This is attributed to the smaller number of weight updates obtained for epoch updating.

We have visualized the movement of the hyperplanes in the input space for ICET. Figure 3.12 illustrates the position of the three hyperplanes after reaching the termination criteria for each respective subset of increasing complexity. For the first subset (points from clusters $A\&B$), all three hyperplanes move quickly in a position where clusters A and B can roughly be separated (after 1692 updates). The RMS criterion of 0.65 prevented the network from learning a local solution, where the two clusters would have been clearly separated by the hyperplanes. For the second subset (points from clusters $A\&B\&C$), one hyperplane moved into a better position to separate clusters A and B, whereas the other two hyperplanes moved into a position to separate clusters A and C (after 6092 updates and a RMS termination value of 0.45). For the final subset, two hyperplanes only improved their respective positions for separating between clusters $A\&B$ and $A\&C$, whereas the third hyperplane moved into a position to separate clusters A and D (after 22472 updates and RMS termination value of 0.20).

We have also visualized the movement of particular weights in a selective weight space (the weights and bias for one hidden unit) for each training strategy. Figure 3.13 shows how the particular weights wander about in the weight space for FSET before obtaining the desired value (indicated in the figure by the point connected to the axial planes). Figure 3.14 illustrates how ICET reduces the unnecessary, inefficient movement of the weights (pictured in Figure 3.13),

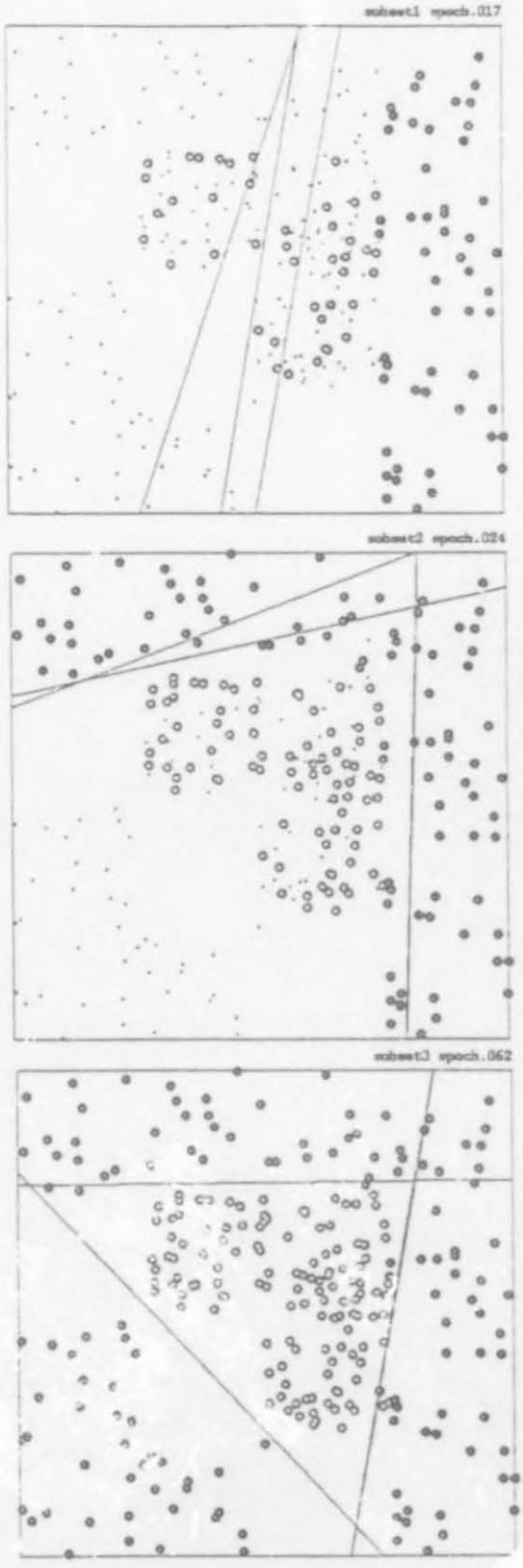


Figure 3.12: Visualization of hyperplanes and input patterns in the input space after reaching the termination criteria for each subset of ICET – Cluster Detection

which is due to multiple solutions. The movement of the weights for the three subsets of increasing complexity is clearly visible. Tightly spaced dots indicate slower movement in the weight space, whereas sparsely spaced dots indicate faster movement. Note the initial faster movement of the weights for the first two easier subsets and the slower movement for the more complex final subset. Figure B.6 shows how CSET also lessens the wandering about in the weight space, exhibited by FSET.

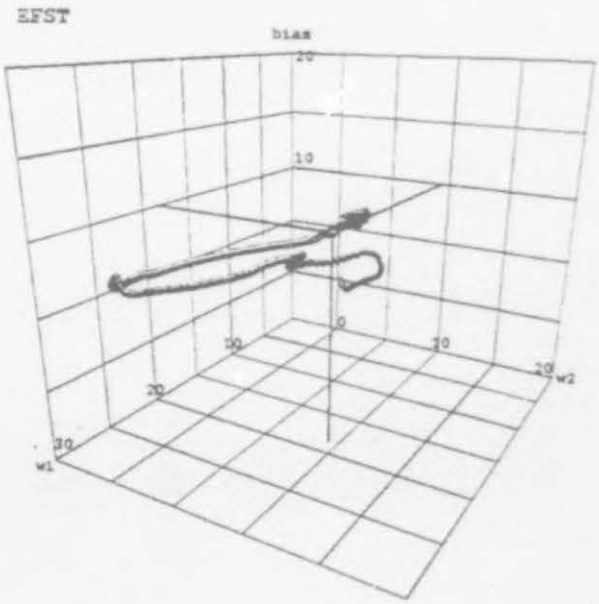


Figure 3.13: Visualization of the movement of weights in the weight space during Epoch FST for Cluster Detection

3.2.12 Comparison with Incremental Learning

In this section we compare Incremental Learning (IL) [Elman, 1991] with Increased Complexity Training and show that, although both organize the training set into sets of increasing complexity, they *differ substantially* in several of ways. IL was discovered after all our training strategies were developed and ICT was reported in [Cloete & Ludik, 1993]. Elman (1991) trained a simple recurrent network on sentences which were generated by a grammar that allow multiple embeddings in the form of relative clauses. The network was trained on 5 passes through each of the following IL phases: (1) in the first phase the training input consisted of 10000 simple sentences; (2) in the next phase the network was trained on 7500 simple and 2500 complex sentences; (3) the training input consisted of a mixture of 5000 simple and 5000 complex sentences; (4) the training set consisted of 2500 simple and 7500 complex sentences; and (5) the network was trained on 10000 complex sentences. The first difference between the two training strategies

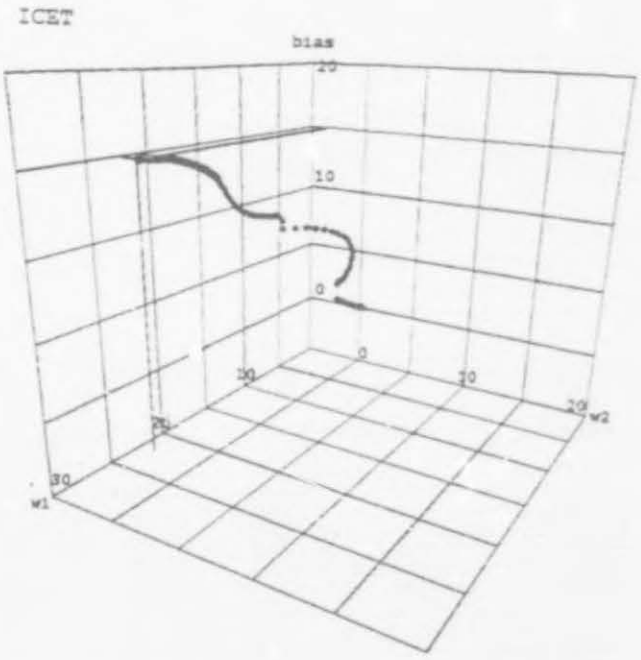


Figure 3.14: Visualization of the movement of weights in the weight space during ICET for Cluster Detection

occur in the way the training input is organized into sets of increasing complexity. IL's subsets have the *same, fixed length* and are not growing in size, whereas ICT's subsets grow in size as well as complexity. Although the complexity levels of training input could not always be determined precisely and it differs from application to application, ICT for our applications has more complexity levels (for Addition and Cluster detection, there are 3 levels, and for Counting, there are 4 levels) compared to IL's case where only two were determined. We conjecture that more complexity levels can be determined from IL's grammar learning training input. For example, simple, medium, and complex sentences, where only simple and complex were used.

Another difference occurs in the training of the subsets itself. In the IL's case each subset was trained with the *same, fixed amount of epochs*. With ICT, training ocurs on combined larger and more complex subsets, each one having a termination criterion. We have investigated various schedules of RMS error values as termination criteria for each subset. Therefore our approach is much more flexible in that it does not use the same fixed amount of epochs for every subset. Instead, we start with a higher RMS error criterion for the first simple subset, and then decrease this value progressively for the subsequent subsets. The amount of epochs trained is therefore dependent on the RMS termination criterion used. For example, with IL each subset (same fixed length) is trained on five epochs in a grammar learning experiment. For Addition, we had two termination criteria, a RMS error value of smaller or equal to 0.15 and a success percentage on the independent test set of greater or equal to 97%, as well as various schedules of RMS values:

subset 1 (0.40-0.70), subset 2 (0.30-0.60), and subset 3 (0.15). This was also the case for the two other applications, Counting and Cluster detection.

We have not only tested ICT with the Elman ASRNN, but altogether with six different ASRNNs as well as feedforward networks, and in all cases obtained superior results for ICT. It was also compared with conventional training strategies using three different applications. We have also extended ICT to Incremental Increased Complexity Training, which should not be confused with Incremental Learning, since it differs even more fundamentally as we show in the following section. We suggest that ICT and IL be regarded as different training strategies, since ICT employs a flexible termination criterion for each subset, whereas IL has none. Also, ICT's subsets are combined larger and more complex, whereas Incremental Learning uses only subsets of the same fixed length.

3.3 Incremental Training Strategies

In this section we elaborate on the two new incremental training strategies, *Incremental Subset Training* and *Incremental Increased Complexity Training* (proposed in section 3.1). IST also caters for the fact that not all training sets can be partitioned into increased complexity subsets. These strategies are compared with ICT, CST, and FST for ASRNNs and Feedforward BP networks using the Addition and Cluster detection applications. A theoretical lower bound is given for the least number of updates when following an incremental training schedule. For the correct increment in subset size the number of updates required for Addition almost attained the predicted lower bound.

The training schedules of ICT and CST leave room for much variation in the size of the initial subset, the increment in subset size, termination criteria for each subsequent subset, and whether training can be terminated earlier without learning on the entire fixed set. This last point is desirable since good generalization may be obtained on a subset of the fixed set. We therefore investigated the following incremental training schedule, also illustrated in Figure 3.15: As success criterion on a subset of examples the RMS error value is used to take into account that subset sizes increase. The user specifies the desired final RMS error value (*RMSdesired* in Figure 3.15) which signifies successful training, as well as the increment in subset size (*SUBincrement*), i.e. the number of single or temporal patterns (for respectively feedforward networks and ASRNNs) to be randomly added to the training subsets. If an independent test set is available, the user specifies the percentage of test examples which should be correctly given by the fully

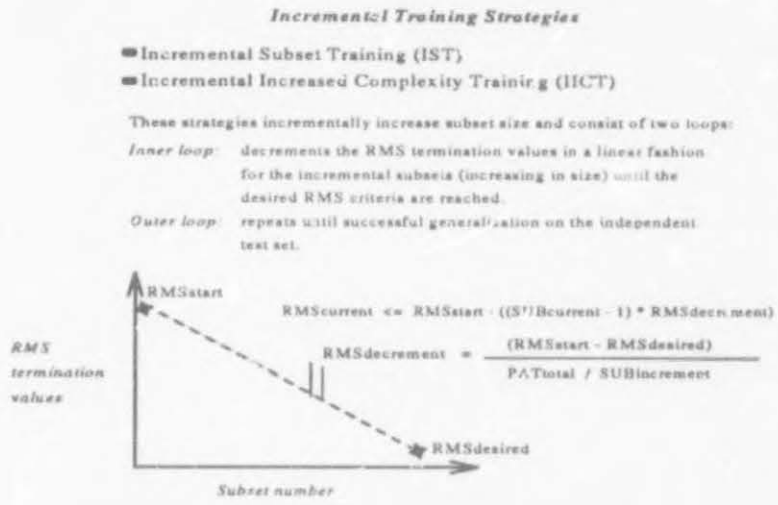


Figure 3.15: The operation of Incremental Training Strategies

trained network in feedforward mode. This allows that training be terminated as soon as the network has generalized sufficiently. The initial subset size is the same as the subset increment. The training schedule then consists of two nested loops: The outer loop repeats until successful generalization is obtained on the test set, while the inner loop repeats training on the subset until the required RMS error value for that training subset is obtained. The RMS termination values for each training subset is decremented linearly (by *RMSdecrement*) from that obtained for the initial subset (*RMSstart*) to the final user-specified value (*RMSdesired*) divided by the number of RMS decrements, thus gradually requiring stricter termination criteria. This prevents training to a too high accuracy on a small set of examples, which may cause an inferior solution to be selected instead of generalizing sufficiently. When the desired RMS value for a particular subset has been reached, the next subset is constructed, the new RMS value is computed and the process repeated. The number of incremental subsets (and number of RMS decrements) is $S = \left\lceil \frac{\text{Number of training patterns in fixed set } (PATtotal)}{\text{Subset increment } (SUBincrement)} \right\rceil$. For ASRNNs *PATtotal* is the number of temporal training patterns in the fixed set, whereas for feedforward networks it is the number of single patterns. In Figure 3.15, *RMScurrent* denotes the current RMS error value obtained by the network after presenting the current subset (*SUBcurrent*). The desired RMS value for the current subset is *RMSstart* - ((*SUBcurrent* - 1) * *RMSdecrement*). IST constructs subsequent incremental training subsets by randomly adding training patterns, whereas IICT constructs the subsets by adding patterns successively from the complexity ranked subsets in multiples of the increment.

To obtain a theoretical lower bound for the number of weight updates with feedforward networks

and ASRNNs, let us denote the subset increment for single and temporal patterns respectively by I_s and I_t. Let us further assume that weights are updated after every single pattern. If training is not repeated on any subset when single patterns are used, then the lower bound for the number of weight updates on the entire training set is $I_s S(S+1)/2$ since the incremental subset sizes are $I_s, 2I_s, \ldots, SI_s$. However, when the training set consists of temporal patterns and the same assumptions hold, the lower bound for the number of updates on the entire training set is $LI_t S(S+1)/2$, where L denotes the average length of the temporal patterns and the incremental subset sizes are $I_t, 2I_t, \ldots, SI_t$. In the following section we show how the lower bound formula for temporal patterns could be used to predict the lowest number of updates when only a portion of the incremental subsets are needed to reach the desired termination criteria.

In sections 3.3.1 to 3.3.3 the Elman, TA and Jordan ASRNNs are evaluated with the difficult Addition application for the incremental training strategies. Section 3.3.4 shows that these strategies are also effective for a feedforward network using the Cluster detection application.

3.3.1 Elman ASRNN : ADDITION

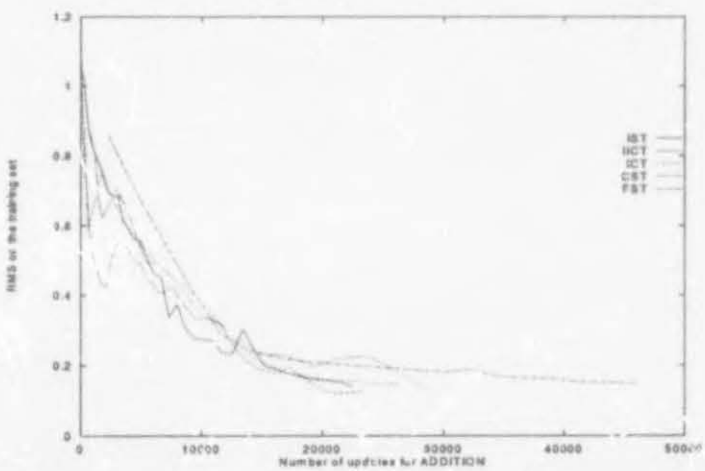


Figure 3.16: The performance of the Incremental Training Strategies for *Addition* with the Elman ASRNN

The incremental training strategies were first evaluated with the Elman ASRNN for the Addition task. The same network as presented in section 3.2.8 with identical training sets and parameters were used. Experiments were repeated with increments of 1, 2, 4, 8, 16 and 32 temporal patterns with corresponding linear RMS decrements. Figure 3.16 shows the performance of the incremental training strategies with the same initial conditions. IST (with optimum increment of four temporal patterns) improved performance by 53% compared to the FST (48300), by 21%

compared to CST (28450), and by 14% compared to ICT (26140). IICT (again with optimum increment of four) improved performance by 52% compared to the FST, by 18% compared to CST, and 11% compared to ICT. The results of ten simulations for each training strategy with the same initial conditions, except for different initial weights, are summarized in Table 3.11.

Training Strategies	Average # updates	Std. D. # updates	Average $\sum E_A$	Std. D. $\sum E_A$	Best # updates	% improvement compared to FST
IICT	32703	11951	1291.52	375.75	16292	28.7%
IST	34700	11220	1301.67	268.38	22582	24.4%
ICT	36186	9388	1324.15	253.46	22735	21.1%
CST	39478	20579	1367.41	279.77	25726	13.9%
FST	45870	16312	1540.02	380.65	29965	-

Table 3.11: A comparison of the average *Addition* simulation results for the Elman ASRNN

From Table 3.11 it is apparent that IICT outperforms FST, CST, and ICT on average by respectively 28.7%, 17.2% and 9.6%, with IICT accordingly obtaining the best number of updates (16292). IST improved the average number of updates compared to FST, CST, and ICT by respectively 24.4%, 12.1%, and 4.1%.

Since the lower bound for the number of updates on the total number of subsets S is $L\ I_t\ S\ (\ S\ +\ 1\ )\ /\ 2$, the corresponding lower bound for the number of updates on a portion of the subsets, say S_p, is $L\ I_t\ S_p\ (\ S_p\ +\ 1\ )\ /\ 2$. For increments 1, 2, 4 and 8 the number of updates required almost attained the optimum lower bound. The number of updates required for IST (with increment four) was 22582, whereas the calculated lower bound was 22464 for an average temporal pattern length of 7.2, and requiring only 156 of the 320 temporal patterns (39 subsets out of a possible 80). When this set of 156 patterns were used for FST, 45360 updates were necessary compared to the 22582 of IST to also reach 97% success on the test set. For increments 16 and 32 more updates were required due to the repetition of their final subsets to reach the desired RMS criteria and generalization. Table 3.12 contains the detailed results for IST and FST.

For IICT we have also investigated the distribution of temporal patterns from each complexity class. For the experiments increments of 1, 2, 4, 8, 16 and 32 temporal patterns on four different fixed sets numbered A to D were used. The distribution of temporal patterns of 1-, 2- and 3-column additions were 64:96:160 for set A, 32:96:192 for B, 16:64:240 for C, and 39:122:159 for D. An increment of 4 temporal patterns in all experiments produced the least number of weight

IST: I_t	S	Updates	S_p	Patterns	Lower Bound
1	320	141631	197	197	141122
2	160	23075	56	112	22982
4	80	22582	39	156	22464
8	40	36620	35	280	36288
16	20	28975	20	320	24192
32	10	61420	10	320	12672
FST	1	48405	1	320	—

Table 3.12: Training results for IST and FST

updates, respectively 23340 for set A, 24856 for B, 27851 for C, and 26871 for D. The error curve for fixed set A is compared in Figure 3.16 for the various training strategies. An increment of 2 temporal patterns performed second best for sets A, B and C, while the 8 increment was second best for D. From these results we conjecture that the distribution of temporal patterns from each complexity class should not assign too small importance to the less complex classes, as does set C. However, exactly how the temporal patterns should be distributed must be further investigated. We simply note that the 4368 different temporal patterns for 1- to 3-column addition are dominated by additions of 3 columns, therefore if patterns were chosen randomly according to this distribution the effect of increased complexity ordering would be nullified.

The erratic behaviour of the error curve for BP learning prompted the investigation leading to the conclusion that BP is sensitive to initial conditions [Kolen & Pollack, 1991]. The proposed incremental training strategies reduce the effect of many attractor basins which cause the erratic behaviour of the error curve, thus leading to faster convergence. This faster convergence can be visualized in Figure 3.17(a), where the weight values of a particular weight during training (same simulation as in Figure 3.16) are pictured for IST, IICT, and FST. Note how quickly IST and IICT's weight values stabilize as opposed to that of FST. Figures 3.17(b) and (c) show the weight changes of the same weight during the first 1000 updates of training for IICT and FST. The IICT graph show initially larger weight changes when compared to the FST graph, before quickly stabilizing as opposed to the continuous weight changing of FST. The Euclidean distance of weight changes for each strategy (see Figure B.7), further illustrates the faster convergence for IST and IICT. Similar visualizations with other weights were also done, all confirming the above-mentioned results.

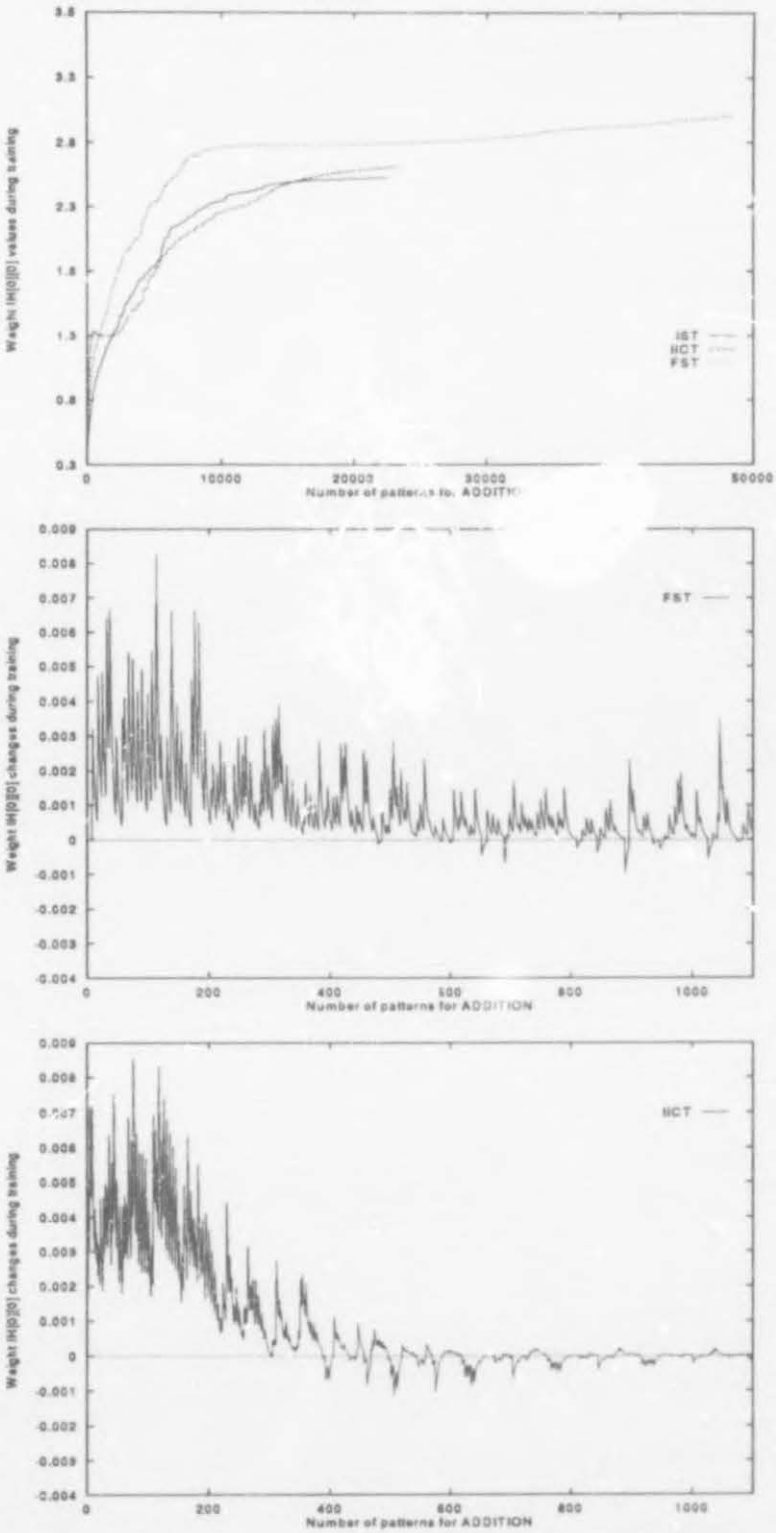


Figure 3.17: (a) Visualization of a weight's values for IST, IICT, and FST (Addition). (b)&(c) Visualization of a weight changes of a particular weight for FST and IICT.

3.3.2 Temporal Autoassociation ASRNN : ADDITION

The Addition experiment with the incremental training strategies was repeated for the Temporal Autoassociation ASRNN with identical architecture, training sets and parameters as were presented in section 3.2.9. The simulation results in Figure B.4 showed, after 70000 weight updates, that the addition experiment for the Temporal Autoassociation ASRNN never converged, no matter what training strategy was used. From the figure it is evident that the two incremental training strategies performed the most respectable under the circumstances. We note that for IICT, the ASRNN initially tried to learn the earlier less complex subsets by decreasing its RMS value from 1.19 to a very low 0.17, before it eventually stabilized around 0.63. For IST the TA ASRNN lowered its RMS value very slowly, before stabilizing around the lowest value, compared to the other training strategies, of 0.53. We again assert that the TA ASRNN is not a suitable architecture for the addition problem, since the latter requires a many to one mapping to be learned, whereas the TA network demands unique internal representations for the different inputs.

3.3.3 Jordan ASRNN : ADDITION

IST and IICT were also evaluated for the Jordan ASRNN with identical architecture, training sets and parameters used for the addition task as presented in section 3.2.10. The optimum subset increment obtained for IICT and IST were respectively four and six. The simulation results of ten simulations for each strategy are summarized in Table 3.13, with the average and standard deviation of the number of updates and the sum of the Euclidean distance of all the weight changes determined.

Training Strategies	Average # updates	Std. D. # updates	Average $\sum E_A$	Std. D. $\sum E_A$	Best # updates	% improvement compared to FST
IICT	52044	39319	1254.54	523.70	17194	61.8%
IST	128416	20098	2655.43	284.04	96007	5.7%
ICT	122698	29131	2431.13	402.71	67591	9.9%
CST	135242	17248	2697.54	234.50	113552	0.7%
FST	136226	18678	2785.21	351.84	103725	-

Table 3.13: A comparison of the average *addition* simulation results for the Jordan ASRNN

Table 3.10 shows IICT performing remarkably well on average when compared to FST, CST,

IST, and ICT. It improved the average number of weight updates by respectively 61.8%, 61.5%, 59.5% and 57.6%, but also proved to be less consistent by having the highest standard deviation. The reason for the latter is due to IICT's best and worst number of updates being respectively 17194 and 126534. The importance of training with subsets that increase in complexity is not only emphasized by IICT's performance, but also by the fact that even ICT outperformed IST by 4.5%. When comparing the Elman and Jordan ASRNN's performance for IST and IICT (with the network weights taken into account), we obtain respective improvements for the Elman ASRNN of 57.1% and only 0.2%, further accentuating IICT's excellent performance for the Jordan ASRNN. The graph in Figure 3.18 illustrates the results of a particular simulation where IICT improved performance when compared to FST, CST, ICT, and IST by respectively 82.8%, 79.5%, 72.7%, and 70.7%. Notice the jumps on IICT's learning curve, which correspond to the incremental addition of more complex temporal patterns, as opposed to the smooth and slow exponential nature of the curves for the other training strategies.

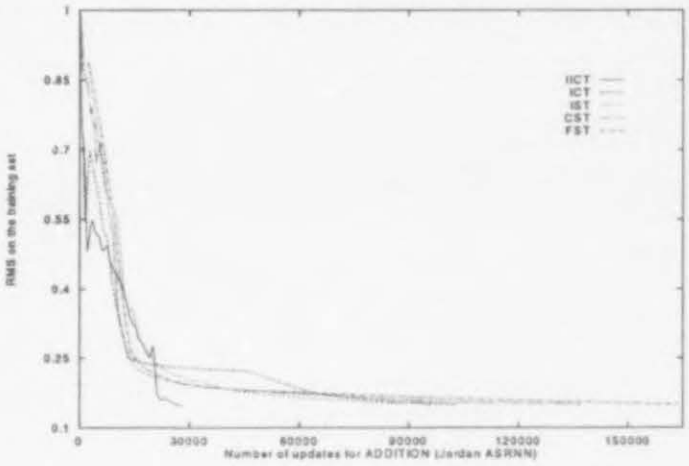


Figure 3.18: The performance of the Incremental Training Strategies for *Addition* with the Jordan ASRNN

3.3.4 Feedforward BP : Cluster Detection

The incremental training strategies were also evaluated for the Cluster detection task using a feedforward BP network with identical architecture, training sets and parameters as were presented in section 3.2.11. Again epoch updating was also evaluated, and the optimum subset increment obtained for IST, ISET, IICT, IICET were respectively two, two, one, and six. The average simulation results of ten simulations for IST and IICT (compared with ICT, CST and FST) are summarized in Table 3.14.

Training Strategies	Average # updates	Std. D. # updates	Average $\sum E_A$	Std. D. $\sum E_A$	Best # updates	% improvement compared to FST
IICT	30864	3108	721.38	76.04	26996	45.4%
IST	31652	4073	758.64	53.15	29446	44.0%
ICT	30199	3945	712.81	76.98	25314	46.6%
CST	34395	4206	798.36	130.04	26734	38.2%
FST	56550	19339	1214.68	390.49	31200	-

Table 3.14: A comparison of the average Cluster detection simulation results for the Feedforward BP network

From Table 3.14 it appears that the average performance of the incremental training strategies are on par with that of ICT, and also outperformed FST and CST: IICT improved performance by respectively 45.4% and 10.3%, and IST by 44% and 8%. IICT's average number of updates is the most consistent by having the lowest standard deviation (3108).

Training Strategies	Average # patterns	Std. D. # patterns	Average $\sum E_A$	Std. D. $\sum E_A$	Best # updates	% improvement compared to FST
IICET	26286	6541	76.26	11.46	21365	53.5%
ISET	32071	17005	63.91	9.43	20020	43.3%
ICET	22035	10783	63.23	11.59	14526	61.0%
CSET	36909	17606	64.35	9.69	20296	34.7%
FSET	38740	7920	72.40	7.60	29900	31.5%

Table 3.15: A comparison of the average Cluster detection simulation results for the Feedforward BP network – epoch updating

From Table 3.15 it is evident that better results are obtained with epoch updating when the patterns are ordered in an increased complexity fashion as opposed to randomly ordered. For example, ICET and IICET, improved performance compared to ICT and IICT by respectively 27% and 14.8%, whereas CST and IST outperformed CSET and ISET by respectively 6.8% and 1.3%.

For IICET and ISET, we have also visualized the movement of particular weights in a weight space consisting of the weights and bias for one hidden unit. Figure 3.19 illustrates how IICET reduces the inefficient movement of the weight in the FSET weight space (pictured in Figure 3.13). Note that for the earlier less complex temporal patterns the movement of the weight

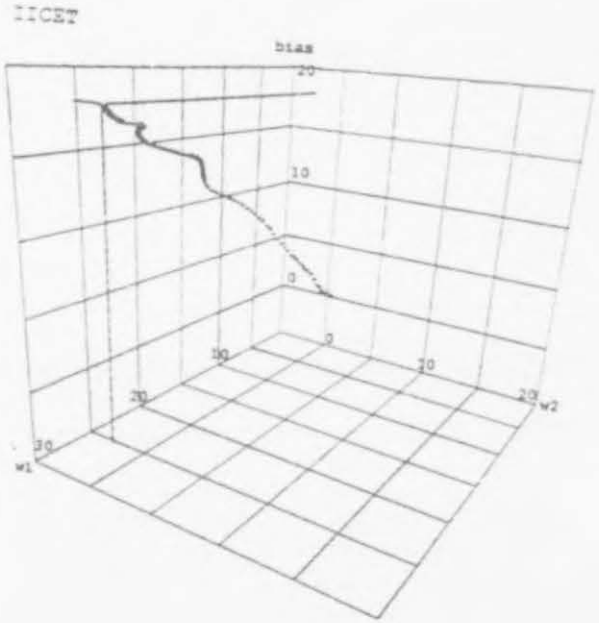


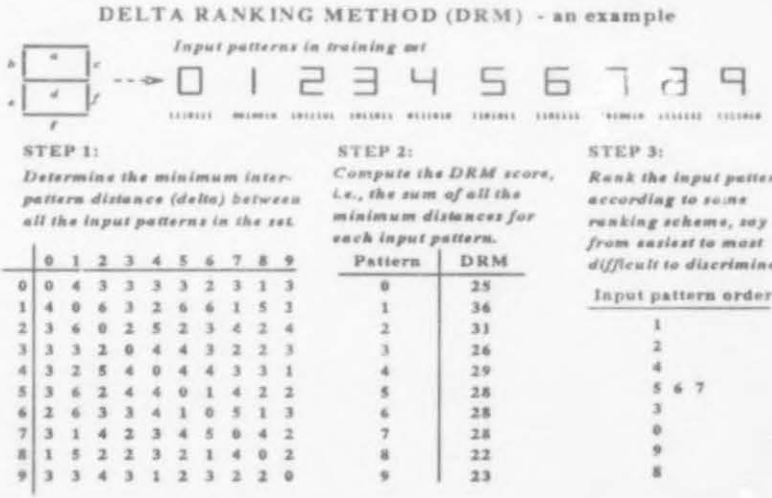
Figure 3.19: Visualization of the movement of weights in the weight space during IICET for Cluster detection

is fast (indicated by sparsely spaced dots) and straight in the direction of the destined weight value. A similar visualization is illustrated for ISET in Figure B.8.

3.4 Delta Training Strategies

In this section we describe and evaluate a group of *Delta Training Strategies*, which is developed independently of the problem's complexity, since the order of its training patterns is based on inter-pattern distance, denoted by *delta*. Any suitable distance metric, such as Hamming or Euclidean distance, can be used. A pattern, in this context, can be a single or temporal input pattern. We employ a method, called the *Delta Ranking Method* (DRM), which determine the complexity relation between the input patterns by obtaining their inter-pattern distances and then ranking them according to some scheme. We explain the method by using the example illustrated in Figure 3.20 where each one of the digits 0...9 is displayed as a seven-bit single input pattern. The first step is to determine the minimum distances between all the input patterns in the set. For example, the minimum distance between the digits 2 and 4 is five, since their input patterns differ by five bits. We then obtain a square symmetric matrix (with diagonal coefficients of zero) which consists of delta values as illustrated in the figure. The second step is to compute the *DRM score*, which is the sum of all the minimum distances for each input pattern. This is achieved by adding up the delta values in each row or column. The third step

is then to rank the input patterns according to some ranking scheme, such as from easiest to most difficult to discriminate. For example, since we have obtained the largest sum of minimum distances (36) for digit 1, it implies that this input pattern is the easiest to discern.



	0	1	2	3	4	5	6	7	8	9
0	0	4	3	3	3	3	2	3	1	3
1	4	0	6	3	2	6	6	1	5	3
2	3	6	0	2	5	2	3	4	2	4
3	3	3	2	0	4	4	3	2	2	3
4	3	2	5	4	0	4	4	3	3	1
5	3	6	2	4	4	0	1	4	2	2
6	2	6	3	3	4	1	0	5	1	3
7	3	1	4	2	3	4	5	0	4	2
8	1	5	2	2	3	2	1	4	0	2
9	3	3	4	3	1	2	3	2	2	0

Pattern	DRM
0	25
1	36
2	31
3	26
4	29
5	28
6	28
7	28
8	22
9	23

Input pattern order
1
2
4
5 6 7
3
0
9
8

Figure 3.20: Illustration of Delta Ranking Method for Delta Training Stategies

We have investigated three different orders of presenting the training patterns using the Delta Ranking Method. The first training strategy, called *Smallest Delta Subset Training* (SDST), orders the input patterns according the smallest DRM score between input patterns, i.e. patterns are ordered from the most difficult to the easiest to discriminate. Figure 3.21 shows an example of subset construction for SDST where the initial subset contains the most difficult pattern to discern (digit 8) and the easiest pattern (digit 1) is only presented as part of the final subset. The next strategy, *Largest Delta Subset Training* (LDST), involves an order presentation of patterns with largest DRM score presented first, i.e. patterns are ordered from the easiest to the most difficult to discriminate. The final strategy is called the *Alternating Delta Subset Training* (ADST), where the first pattern has the smallest DRM score, the second pattern the largest DRM score, third pattern the second smallest, etc. Figure 3.21 also illustrates examples of subset construction for LDST and ADST.

We have further investigated the incremental extensions of SDST, LDST, and ADST, which employ the same training schedule as IST and IICT, and respectively led to *Incremental SDST* (ISDST), *Incremental LDST* (ILDST), and *Incremental ADST* (IADST). These incremental delta training strategies construct the subsets by adding patterns successively from the DRM score ranked subsets in multiples of the increment and according to one of the proposed ranking schemes. For feedforward networks we have also evaluated the epoch updating versions of all

	SDST (Smallest Delta Subset Training)	LDST (Largest Delta Subset Training)	ADST (Alternating Delta Subset Training)
initial subset	8	1	
.	89	12	18
.	890	124	182
.	8903	1245	1829
.	89035	12456	18294
.	890356	124567	182940
.	8903567	1245673	1829405
.	89035674	12456730	18294053
.	890356742	124567309	182940536
final subset	8903567421	1245673098	1829405367

Figure 3.21: Illustration of subset construction for SDST, LDST, and ADST

six these strategies.

The Delta Ranking Method can also be applied to temporal patterns, where a DRM score is then computed for each different temporal pattern in the training set. We illustrate such a case in a subsequent section when the delta training strategies are evaluated for the Elman ASRNN using the Temporal Letter Recognition application. The DRM can in general be applied to any set of n-dimensional real valued input patterns, since the basic computation only involves determining the minimum distance between a point $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n)$. Assume we have N training input patterns. The computational complexity for the first step of DRM is then $N(N-1)/2$, since there are $1 + 2 + \ldots + (N-1)$ computations in determining the minimum distances between the N input patterns. The complexity of computing the second step is $N(N-1)$, since there are N input patterns, for which a DRM score, consisting of $N-1$ additions, must be computed.

In sections 3.4.1 and 3.4.2 the delta training strategies are evaluated with the Elman ASRNN using respectively a sequence generation (Digital Morse Code generation) and sequence recognition application (Temporal Letter recognition). Section 3.4.3 shows that these strategies can also be succesfully applied to feedforward networks.

3.4.1 Elman ASRNN : Digital Morse Code Generation

We first evaluated the delta training strategies for the Elman ASRNN using the Digital Morse Code Generation application, where the task is to sequentially generate the Morse code sequence

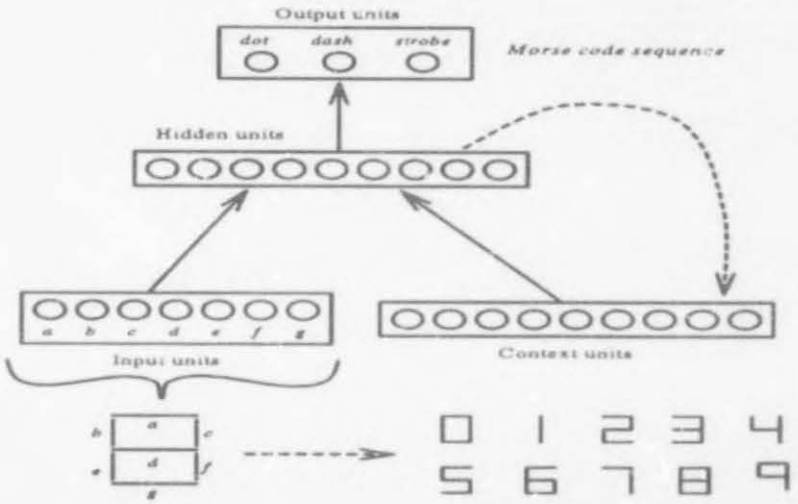


Figure 3.22: Elman ASRNN used for the Digital Morse Code Generation application

for each of the 10 decimal digits, presented as a seven-bit input pattern. The network, illustrated in Figure 3.22, has seven input units, nine hidden and nine context units (optimum number experimentally determined), and three binary outputs respectively indicating *dot*, *dash* and a *done* bit which is on only when the last dot or dash is output.

For successful training a RMS error value of less that 0.15 for the training set and a success ratio greater than 98% on the test set were required. For SDST, LDST, and ADST four subsets were constructed according to the ranking schemes illustrated in Figure 3.21. The optimum RMS termination values for these three training strategies were respectively, $0.75-0.65-0.55-0.15$, $0.55-0.50-0.40-0.15$, and $0.65-0.55-0.45-0.15$. These values make sense, since they were less strict to accomodate the more complex subsets of SDST, very strict for the easier subsets of LDST, and in between for the alternating patterns of ADST. The optimum RMS termination values for CST were $0.65-0.55-0.45-0.15$. The optimum subset increment obtained for ISDST, ILDST, IADST, and IST were respectively two, three, five, and one. The average simulation results of ten simulations for each training strategy are summarized in Table 3.16.

From Table 3.16 ILDST and ISDST outperformed all the other training strategies on average, including IST. ILDST compared to FST, CST, SDST, LDST, IST, and ISDST yielded improvements of 58.1%, 55.7%, 53.1%, 41.7%, 32.1%, and 5.4%. ISDST achieved the lowest average sum of Euclidean distances of all the weight changes (1124.10) as well as the best number of updates (12690). LDST and SDST also proved useful training strategies, outperforming the conventional strategies, CST (24% and 6%) and FST (28.1% and 11%). Only ADST performed worse than FST for this particular application. Since the Alternating Delta Training Strategies performed

Training Strategies	Average # updates	Std. D. # updates	Average $\sum E_A$	Std. D. $\sum E_A$	Best # updates	% improvement compared to FST
ILDST	30870	9731	1154.91	633.87	15465	58.1%
ISDST	32646	20769	1124.10	334.61	12690	55.7%
IADST	71682	24252	2062.30	810.82	28450	2.7%
IST	45340	25922	1530.75	610.51	16950	38.5%
SDST	65580	31182	1863.34	628.04	24600	11.0%
LDST	52960	29399	1785.57	923.57	14503	28.1%
ADST	89380	4860	2456.08	588.82	74800	-
CST	69730	26692	1860.15	463.52	29650	5.4%
FST	73700	27398	2353.44	925.27	14500	-

Table 3.16: A comparison of the average *Digital Morse Code Generation* simulation results for the Elman ASRNN

very well for applications without any temporal *output* sequences (see the next two sections) and poor for this application, we can only deduce that the change in complexity from pattern to pattern for these type of tasks in an Elman ASRNN should be gradual and not contrasting. However, for a Jordan ASRNN, which specializes in tasks that require sequence production, this alternating presentation of input patterns would not necessary be a problem. Table B.3 presents the results for a particular simulation, whereas Figure 3.23 highlights only the Incremental Delta Training Strategies results compared to FST: ISDST yielding an improvement of 76.3%, followed by ILDST with 71.1%, and IADST with 27.7%.

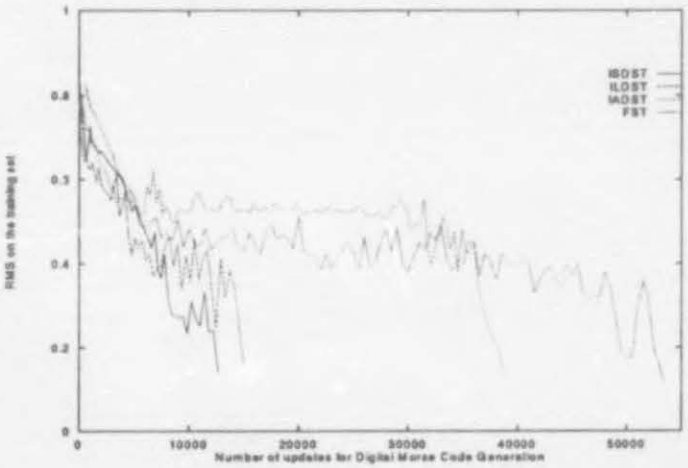


Figure 3.23: Performance of the Incremental Delta Training Strategies compared to FST for the *Digital Morse Code Generation*

The optimum increment in patterns for ILDST, ISDST, and IST were small (respectively 3, 2, and 1), suggesting in general that small increments perform better. This is supported by the observation that single pattern updating rather than epoch updating usually causes faster convergence for ASRNNs. We conjecture that the Incremental Delta Training strategies (which start with a small set of examples) allow weights to find a good position in weight space quickly, thus requiring less weight movement than that suggested when the entire training set is presented at once. Figures 3.24 illustrates this fact by comparing the weight changes of the same weight for ISDST, ILDST, and FST. The ISDST graph shows initial large weight changes before rapidly stabilizing, indicating that a good position has been found quickly in weight space. In contrast, the FST graph shows large weight changes continually. Ordering of training patterns according to our proposed ranking schemes, especially when presented in an incremental fashion, thus forced the network to discriminate between classes early in the training process, leading to reduced training time.

3.4.2 Elman ASRNN : Temporal Letter Recognition

In this section the Delta Ranking Method is applied to temporal input patterns for the Elman ASRNN using the Temporal Letter Recognition application, where the task is to recognize temporal letter sequences, each consisting of seven single input patterns which represents a a letter from A to J. The network has two input units, one receiving one bit of the temporal sequence at a time and one done bit, which indicates when the last input pattern of the sequence is presented. Six hidden and six context units proved experimentally to be sufficient. Four output units represented a binary coding corresponding to each of the temporal letter sequences.

We have applied the Delta Ranking Method to the set of temporal letter sequences: A square symmetric matrix was obtained containing the minimum distances between all the temporal input patterns; the DRM score for each different temporal pattern was then computed. For a ranking scheme of easiest to most difficult to discriminate, the temporal letter sequences were ordered as follows: I, J, C, F, E, H, D, A, G, B. The RMS termination value for the final subset was 0.28 for the training set and a success percentage of more than 90% on the test set were required. For SDST, LDST, ADST three subsets were constructed according to the three ranking schemes. The optimum RMS termination values for these three training strategies were respectively, 0.45 – 0.40 - 0.28, 0.40 – 0.35 – 0.28, and 0.40 – 0.35 – 0.28. The optimum RMS termination values for CST were 0.45 – 0.35 – 0.28. The optimum subset increment obtained for ISDST, and ILDST was three, whereas it was four for IADST and IST. We note that the

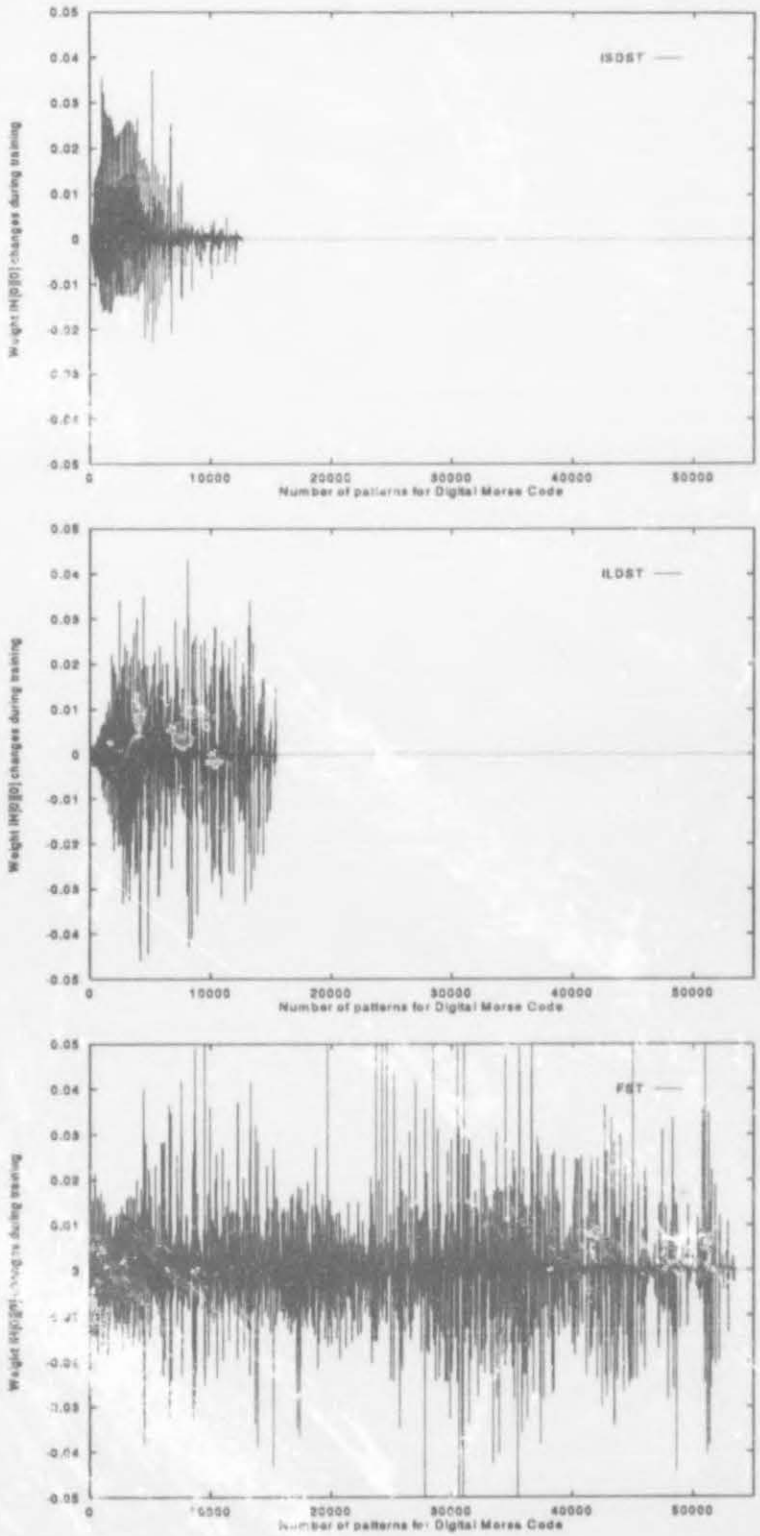


Figure 3.24: Visualization of a weight changes for ISDST, ILDST, and FST (Digital Morse Code Generation)

complexity changing from pattern to pattern is more gradual for the first two strategies, than the last two for which it is contrasting or random, suggesting why similar optimum increments were respectively obtained. The average simulation results are summarized in Table 3.17.

Training Strategies	Average # updates	Std. D. # updates	Average $\sum E_A$	Std. D. $\sum E_A$	Best # updates	% improvement compared to FST
IADST	26690	4410	455.43	80.83	20482	57.8%
ILDST	29040	8589	467.80	126.06	17171	54.0%
ISDST	30601	8117	516.77	105.25	23814	51.6%
IST	33260	6848	582.11	149.92	22820	47.4%
ADST	40390	11983	693.57	200.89	17290	36.1%
SDST	45245	12454	767.55	205.64	25620	28.4%
LDST	51902	19430	975.55	396.05	28350	17.8%
CST	55455	23199	1008.93	460.95	30075	12.2%
FST	63175	28056	1086.22	420.27	24500	-

Table 3.17: A comparison of the average *Temporal Letter Recognition* simulation results for the Elman ASRNN

This time IADST proved to be the best training strategy, improving the average updates compared to FST, CST, LDST, SDST, ADST, IST, ISDST, and ILDST by respectively 57.8%, 51.9%, 48.6%, 41%, 33.9%, 19.8%, 12.8%, and 8.1%. It was also the most consistent by having the lowest standard deviation and achieved the lowest average sum of Euclidean distances of all the weight changes (455.43). Although ILDST obtained the best number of updates (17171), it is interesting to note that a non-incremental delta strategy, ADST, came close with 17290 updates. The very good performance of the alternating delta strategies can be attributed to the network being forced by the alternating patterns to discriminate between contrasting complexity classes early in the training process. For this application, having temporal input patterns, all the delta training strategies outperformed the conventional FST and CST strategies, and all three incremental delta training strategies yielded improvements compared to IST. Figure 3.25 illustrates the results for a particular simulation, highlighting ADST, IADST, ISDST, and ILDST's respective improvements of 79.3%, 71.9%, 70.9%, and 68.1% compared to FST. Note the instability of the FST curve as opposed to the relative smoothness of the error curves of the delta training strategies.

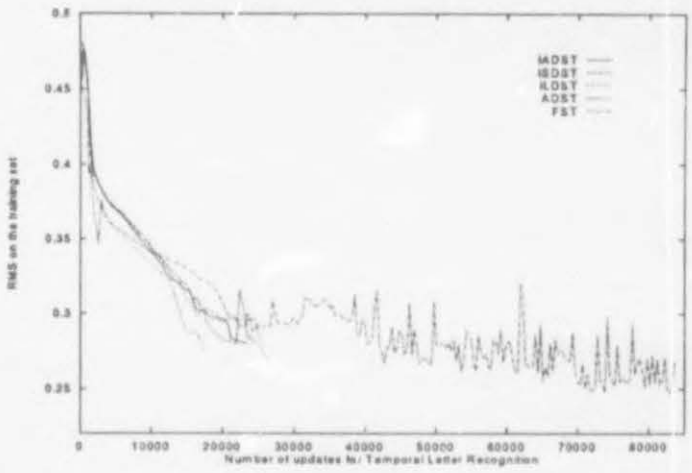


Figure 3.25: Performance of the Incremental Delta Training Strategies compared to FST for the *Temporal Letter Recognition* problem

3.4.3 Feedforward BP : Digit Recognition

The delta training strategies were finally tested on a Digit Recognition application for a feedforward BP network. The task comprises the recognition of binary patterns representing the 10 decimal digits encoded for a seven-segment display. The input patterns were similar to the input of the Digital Morse Code Generation task, except that at most one bit of a pattern was randomly corrupted with 10% probability. The network, illustrated in Figure B.9, has seven input units, ten hidden units (the optimum value experimentally determined), and one real valued output. For successful training a RMS error value of less than 0.185 for the training set and a success percentage greater than 80% on the test set were required. The fixed training set had 349 patterns and the test set 151.

The optimum RMS termination values were again obtained for the four subsets of SDST (SDSET), LDST (LDSET), ADST (ADSET), and CST (CSET). The optimum subset increment obtained for ISDST, ISDSET, ILDST, ILDSET, IADST, IADSET, IST, and ISET were respectively 2, 5, 1, 4, 11, 12, 3, and 5. We note that the optimum subset increment is always larger for the epoch version of a particular training strategy, suggesting that for epoch updating to be effective the subset must rather be larger than smaller. Figure 3.26 shows the three incremental epoch delta training strategies, ISDSET, ILDSET, and IADSET for a particular simulation compared to FST, yielding respective improvements of 52.1%, 23.1%, and 22.3%. Note that although the RMS values for ILDSET and IADSET were already below the desired 0.185 early during training, both struggled to reach a success ratio greater than 80% on the test set. The

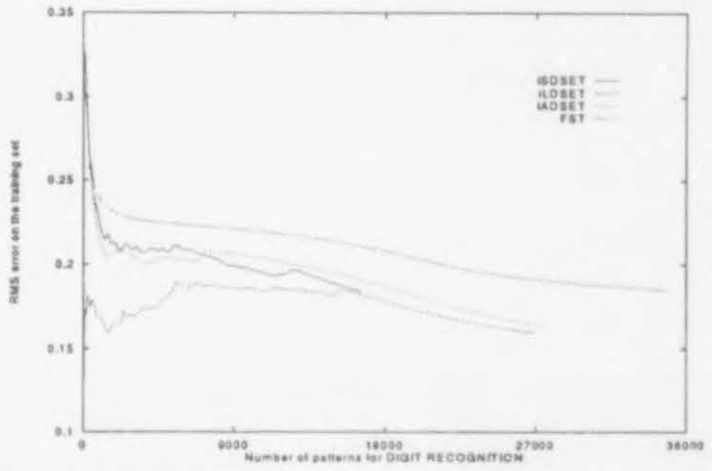


Figure 3.26: Performance of the Incremental Epoch Delta Training Strategies compared to FST for *Digit Recognition*

low RMS values for ILDST are attributed to patterns being ordered from the easiest to the most difficult to discriminate (i.e. those with the largest DRM scores are presented first). Table B.4 gives more detail regarding each training strategy for this simulation. The average simulation results for each non-epoch updating training strategy are summarized in Table 3.18.

Training Strategies	Average # updates	Std. D. # updates	Average $\sum E_A$	Std. D. $\sum E_A$	Best # updates	% improvement compared to FST
ISDST	21274	5968	220.83	63.04	14524	64.5%
IADST	29149	10457	291.12	68.03	19453	51.3%
ILDST	32951	4615	273.33	49.37	24531	45.0%
IST	27463	6290	281.62	60.62	20343	54.1%
ADST	30260	5522	314.57	58.45	19023	49.5%
LDST	33848	4197	337.64	39.52	27217	43.5%
SDST	36562	7438	366.95	79.16	26267	38.9%
CST	53607	11135	529.66	107.44	30254	10.5%
FST	59888	15274	562.93	133.82	33504	-

Table 3.18: A comparison of the average *Digit Recognition* simulation results for the Elman ASRNN

For this application ISDST performed the best among the non-epoch training strategies, improving the average updates compared to FST, CST, LDST, SDST, ADST, IST, IADST, and ILDST by respectively 64.5%, 51.9%, 48.6%, 41%, 33.9%, 19.8%, 12.8%, and 8.1%. It achieved the lowest average sum of Euclidean distances of all the weight changes (220.83) as well as the

best number of updates (14524). The very good performance of the ISDST can be attributed to the good discriminating position in weight space achieved by presenting the network initially with a few complex patterns in an incremental fashion. We note that ADST as a non-incremental delta strategy, obtained the second best number of updates. The non-incremental delta training strategies, ADST and LDST, also came relatively close to the average number of updates of their incremental versions. For this application all the non-epoch delta training strategies outperformed the conventional FST and CST strategies.

Training Strategies	Average # patterns	Std. D. # patterns	Average $\sum E_A$	Std. D. $\sum E_A$	Best # updates	% improvement compared to FST
ISDSET	19926	5688	12.70	2.17	14111	66.7%
ILDSET	27161	4585	12.42	1.24	21943	54.6%
IADSET	34191	10983	14.78	2.39	15675	42.9%
ISET	24694	6110	13.21	1.80	16630	58.8%
ADSET	29664	3933	13.62	0.62	21742	50.5%
LDSET	31998	3274	13.12	0.69	26519	46.6%
SDSET	31301	3605	12.62	0.57	27427	47.7%
CSET	42381	16148	14.45	2.43	25213	29.2%
FSET	49139	20807	13.95	2.65	24081	17.9%

Table 3.19: A comparison of the average *Digit Recognition* simulation results for the Elman ASRNN - epoch updating

From Table 3.19 it is evident that better results are obtained with epoch updating (the results are compared to FST). ISDSET improved performance compared to FST, FSET, CSET, SDSET, LDSET, ADSET, ISET, IADSET, and ILDSET by respectively 66.7%, 59.4%, 53%, 36.3%, 37.7%, 32.8%, 19.3%, 41.7%, and 26.6%. It also obtained the best number of updates, namely 14111. Again the Incremental Delta training strategies performed better than their non-incremental versions, except for ADSET, which obtained a better average number of updates than IADSET. To put this result in a better perspective, we note that IADSET obtained a much higher standard deviation on the average number of updates, as well as an improved best number of updates compared to ADSET (15675 versus 21742). Again all the Delta training strategies proved to be very effective in reducing the training time, especially when compared with the conventional strategies CSET and FSET. Thus the ordering of training patterns according to the proposed schemes, especially in an incremental fashion, allows that a good discriminating position in weight space be the target early in the learning process, yielding faster convergence.

3.5 Summary

In this chapter we have showed that the variation of a training strategy is a viable alternative method to produce an effective solution for training ASRNNs and feedforward networks within a feasible time. We have developed *Increased Complexity Training* (ICT), two *incremental* training strategies (IICT and IST) and six *delta* training strategies (SDST, LDST, ADST, ISDST, ILDST, and IADST), and showed that they lead to considerable improvement in training time and alleviate problems such as a too large training set and a long training time. Training strategies and ASRNNs were evaluated simultaneously for different applications, varying in complexity and ranging from sequence recognition to sequence generation.

It was demonstrated with six different ASRNNs and feedforward networks that for several applications, ICT outperformed the conventional CST and FST. The incremental training strategies (including the incremental versions of the delta strategies) also effectively addressed further problems with the size of the initial subset, the increment in the subset size, the termination criteria for each subsequent subset, and the number of examples required for good generalization. IST and IICT showed for Addition that a good training set can be quite small to provide very good generalization. IST, for example, for a particular simulation needed only half of the training set to double the performance of training on a fixed set. It also improved performance by 50% compared to FST on the same half of the training set. For the correct increment in subset size the number of updates required for addition almost attained the theoretical lower bound.

The delta training strategies employ the Delta Ranking Method, which determines the complexity relation between the input patterns by obtaining their inter-pattern distances and then ranking them according to some scheme. Three basic ranking schemes were introduced that led to the different delta training strategies. Incremental delta training strategies performed the best overall, signalling that the ordering of training patterns according to our proposed delta ranking schemes, especially when presented in an incremental fashion, forces the network to discriminate between classes early in the training process, leading to reduced training time.

The operation of the training strategies were investigated by visualizing the movement of hyperplanes in the input space, the movement of particular weights in the weight space, and the Euclidean sum of weight changes. The visualization graphs showed how the new training strategies efficiently reduce the weight activity and obtain faster convergence.

We have also evaluated Elman, Jordan, TA and other newly constructed ASRNNs for the

Counting and Addition tasks, and found the Output-to-Hidden Hidden-to-Hidden ASRNN to be the best simple recurrent network architecture, outperforming the other ASRNNs between 44% and 87% for the training strategies. This network combines the best features of the Elman and Jordan ASRNN, causing it to be able to learn input and/or output sequences. The Elman, TA, and Output-to-Hidden Hidden-to-Hidden ASRNNs have a context layer on the input level and is therefore equipped to effectively deal with input sequences. They accordingly performed very well for the Counting application with its variable input sequences. The Jordan ASRNN is restricted in not being able to remember input not reflected directly in its output, whereas the TA ASRNN can only deal with tasks that allow learning an inverse mapping of the current input and context units. ASRNNs with output-to-output feedback, such as the Output-to-Output Hidden-to-Hidden and Output-to-Output networks, did not prove to be useful for tasks with temporal input. The next chapter investigates ASRNNs further in terms of their classification capabilities and dynamics.

Chapter 4

Classification Capabilities and Dynamics Analysis

In this chapter the classification capabilities and dynamics of architecture-specific recurrent neural networks are analyzed. The analysis entails complexity issues, such as their classification capabilities, as well as their classification and learning dynamics. By investigating the classification capabilities of ASRNNs, we examine the behaviour of these networks in terms of the number and possible types of regions the network is capable of forming in the input and hidden activation spaces during the classification process. We also explore the dynamics of the classification process by analyzing the internal representation (clusters formed by the hidden unit activation values) of the network. It is shown that an ASRNN can learn to mimic closely a finite state machine (FSM), both in its behaviour and in its state representation. We obtain the internal representation of the network by visualizing the input activation space and using clustering techniques, such as Hierarchical Cluster Analysis, Principal Component Analysis, and the unfamiliar Sammon Transformation Analysis. The labeled clusters obtained by these techniques then represent the states of the simulated FSM. The network dynamics is depicted by the transitions between states. The correspondence between the simulated FSM and the one initially constructed for the Addition training data is then determined. The dynamics of the learning process in ASRNNs is then explored by visualizing the evolution of clusters formed during the learning process. It is also shown that the network's performance improves with larger temporal window sizes.

For the theoretical analysis, we consider networks where the unit nonlinearities are threshold elements with binary outputs, i.e. units with a step threshold activation function (see Appendix

A for a description of threshold activation functions). A *threshold unit* having an n-dimensional input with weight vector $W = (w_1, \ldots, w_n)$ and a threshold ω gives an output one for an input vector $X = (x_1, \ldots, x_n)$ if $W \cdot X - \omega = \sum_j w_j x_j - \omega \geq 0$ and zero if $W \cdot X - \omega < 0$, where $W \cdot X$ is the inner product. The focus is to investigate the capabilities and dynamics of the three main architecture-specific recurrent threshold networks (ASRTNs), namely Elman, Jordan, and Temporal Autoassociation threshold networks. The hidden and output layers of these networks have threshold units, whereas the context units (for Elman or TA ASRTNs) and state units (for Jordan ASRTNs) are linear, similar to the input units. Central to this analysis is that an architecture-specific recurrent threshold network can be viewed as a feedforward threshold network with special additional inputs, the context or state units, whose values are provided by the network itself (specifically by the previous hidden or output units via fixed one-to-one connections). These special context or state inputs, which operate on the same level as the input layer, give the network the capability to store temporal patterns (or to predict the next input). The conclusions of this analysis can be extended to networks with the well-known sigmoid activation function. This can be accomplished by a sigmoid function that approximates a threshold function or just by multiplying all the weights and thresholds in the network by a sufficiently large constant, forcing the outputs to be close to one or zero (the sigmoid function is in the "high-gain limit", i.e. the gain parameter is large).

4.1 Classification Capabilities

We examine the number and possible types of cells (see definition 4.1) in the input and hidden activation space (see section 1.1) for classification as a function of the number of network units. The basic functioning of a multilayer neural network is that the first hidden layer uses hyperplanes to partition the input space into a number of cells. The main function of the additional layers, in particular the output layer, is then to group these cells into larger regions.

Definition 4.1:

A *cell* [Makhoul *et al*, 1989] is a polyhedric region in the input activation space, which are labeled by the hidden activation values that specify on which side of each hyperplane the points in that region lie.

A cell in the input space has a corresponding cell in the hidden space, both which are labeled by the same hidden activation values. Figure 4.1 illustrates, for example, a feedforward threshold network (FFTN) with two cells, labeled 0 and 1, in its input and hidden space. The cells formed

in the input space can be divided into open and closed cells.

Definition 4.2:

An *open cell* in the input space has input variables which are unbounded.

Definition 4.3:

A *closed cell* in the input space has input variables which are always bounded.

Figure 4.2 provides an example of a FFTN's input space having two open cells, labeled 00 and 11, and one closed cell, labeled 10.

Definition 4.4:

An *imaginary cell* is labeled by hidden activation values that have no corresponding input values and appears as a cell in the hidden space, but not in the input space.

The FFTN's hidden space, illustrated in Figure 4.2, contains an imaginary cell, labeled 01, which has no corresponding values in the input space. To distinguish between cells that are formed in the input space and those that are imaginary, we introduce the term *real cells* to denote the former. The number of real cells is the sum of the open and closed cells. When the term *cell* is used on its own, it indicates a real cell.

Definition 4.5:

A region or a set of points is *connected* if it cannot be split into parts that are not adherent to one another, that is, no parts contain at least one neighborhood point of the other.

Definition 4.6:

A set of N vectors is in *general position* in n-dimensional space if every subset of n or fewer vectors is linearly independent [Cover, 1965].

In the following sections we discuss the number and types of cells for Elman, Jordan, and Temporal Autoassociation ASRTNs, compare the results with feedforward threshold networks, illustrate them with a few examples, and interpret the equations obtained.

4.2 Elman ASRTNs

We denote a feedforward threshold network (FFTN) with n input units, a single hidden layer of h threshold units, and s output threshold units by *n:h:s* FFTN. When $s = 1$, the network is denoted by *n:h:1* FFTN.

An Elman architecture-specific recurrent threshold network (ASRTN) with n input units, a single hidden layer of h threshold units, a single context layer of h linear units, each containing a copy of a corresponding hidden threshold unit value on the previous time step, and s output threshold units, is denoted by *n:h:s Elman ASRTN*. When $s = 1$, we denote the network by *n:h:1 Elman ASRTN*. An *n:h:s* Elman ASRTN can be viewed as an *(n+h):h:s* FFTN having h special additional inputs, the context units.

4.2.1 Number of Cells

We assume that the h hyperplanes in the first hidden layer are in general position (definition 4.6). This implies that none of the hyperplanes are parallel to each other (that is, all hyperplanes intersect each other) and no more than n hyperplanes intersect at a point. The purpose of this assumption is to assure that no subset of the hyperplanes degenerates, which is a necessary requirement in obtaining equations for the number of cells. Let $C(h, m)$ be the number of cells formed by h planes in m-dimensional space, where $m = n + h$ is the expanded input dimension $m = n + h$ of an *n:h:s* Elman ASRTN. The number of cells formed by an *n:h:s* FFTN [Makhoul *et al*, 1989] is

$$\begin{aligned} C(h,n) &= C(h-1,n) + C(h-1,n-1) \\ &= \sum_{i=0}^{min(h,n)} \frac{h!}{i!(h-i)!} \\ &= \begin{cases} 2^h & h \leq n \\ \sum_{i=0}^{n} \frac{h!}{i!(h-i)!} & h > n \end{cases} \end{aligned} \tag{4.1}$$

Since $h < m = n + h$, the number of cells formed by an *n:h:s* Elman ASRTN is

$$\begin{aligned} C(h,m) &= \sum_{i=0}^{min(h,m)} \binom{h}{i} \\ &= \sum_{i=0}^{h} \frac{h!}{i!(h-i)!} \\ &= 2^h \end{aligned} \tag{4.2}$$

Thus the number of cells formed by h hyperplanes in m-dimensional space is independent of the expanded input dimension m of an Elman ASRTN.

Adding an input or context unit:

For an Elman ASRTN increasing the input dimension by one implies either adding a unit to the input layer (which has no effect on the number of cells $C(h, m) = 2^h$), or adding a unit to

the context layer (which implies adding a unit to the hidden layer leading to an increase of the number of cells to $2^{(h+1)}$).

Adding a hidden unit:
Adding a hidden unit implies adding a hyperplane, which increases the number of cells to $2^{(h+1)}$, having the effect of doubling the number of cells.

4.2.2 Open and Closed Cells

The cells formed in the input space can be divided into open and closed cells (definitions 4.2 and 4.3). In general $C(h,m) = C_o(h,m) + C_c(h,m)$, where $C_o(h,m)$ is the number of open cells and $C_c(h,m)$ the number of closed cells. The number of open cells formed by an *n:h:s* FFTN [Makhoul *et al*, 1989] is

$$C_o(h,n) = \begin{cases} 2^h & h \leq n \\ 2\sum_{i=0}^{n-1} \frac{(h-1)!}{i!(h-1-i)!} & h > n \end{cases} \tag{4.3}$$

The number of closed cells formed by an *n:h:s* FFTN [Makhoul *et al*, 1989] is

$$C_c(h,n) = \begin{cases} 0 & h \leq n \\ \frac{(h-1)!}{n!(h-1-n)!} & h > n \end{cases} \tag{4.4}$$

For an Elman ASRTN the expanded input dimension m is always larger than the number of hyperplanes h, i.e. $h < m$. Thus applying equations (4.3) and (4.4) to Elman ASRTNs, $C_o(h,m) = 2^h$ and $C_c(h,m) = 0$. Thus for an Elman ASRTN there can be no closed cells and all of them are open.

4.2.3 Imaginary Cells and Disconnected Regions

According to Makhoul *et al* the number of imaginary cells (definition 4.4) for a FFTN is equal to

$$C_i(h,m) = 2^h - C(h,m). \tag{4.5}$$

Since $h < m$ for an Elman ASRTN, $C(h,m) = 2^h$. Thus $C_i(h,m) = 2^h - 2^h = 0$. Therefore Elman ASRTNs cannot have any imaginary cells. Decision regions in the input space consist of connected cells. Makhoul *et al* showed that one hidden layer threshold gates are also capable of forming disconnected decision regions, which are connected (definition 4.5) together through a set of imaginary cells. But Elman ASRTNs do not have any imaginary cells in their input

space; thus it can be concluded that decision regions formed by an Elman ASRTN are either convex decision regions or non-convex but connected regions. Therefore Elman ASRTNs are not capable of forming disconnected decision regions.

4.2.4 Examples

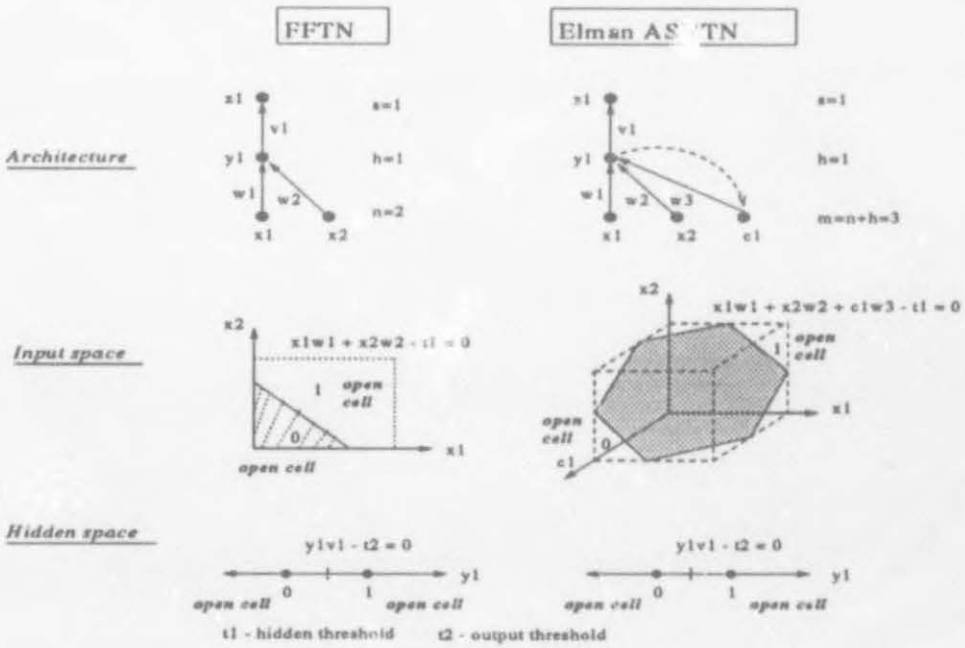


Figure 4.1: Example 1: Architecture, input and hidden space for *2:1:1* FFTN and Elman ASRTN.

In this section we compare *n:h:s* FFTNs with their corresponding *n:h:s* Elman ASRTNs in terms of the number and types of cells formed. We proceed with examples illustrating the two cases, $h \leq n$ and $h > n$.

Example 1: Let $n = 2$, $h = 1$, and $s = 1$. Figure 4.1 illustrates a *2:1:1* FFTN and the corresponding *2:1:1* Elman ASRTN in terms of network architecture, input and hidden space. For convenience, we refer to the Elman ASRTN's *input-context activation space* (the space defined by the input and context unit activations) also as input space. To avoid visualization clutter, only the positive quadrant of the input space is shown. Each cell is labeled with a binary number y_1 that specifies on which side of the hyperplane ($x_1w_1 + x_2w_2 - t_1 = 0$ for the FFTN and $x_1w_1 + x_2w_2 + c_1w_3 - t_1 = 0$ for the ASRTN) the points in that cell lie. In this example the weights and thresholds of the FFTN are $w_1 = 0.9$, $w_2 = 0.8$, $t_1 = 0.6$, $v_1 = 0.5$, and $t_2 = 0.25$. The corresponding values for the ASRTN are $w_1 = 0.3$, $w_2 = 0.35$, $w_3 = 0.4$, $t_1 = 0.5$,

$v_1 = 0.5$, and $t_2 = 0.25$. For the FFTN the number of cells is $C(h,n) = 2^h$, since $h \leq n$. Thus $C(1,2) = 2^1 = 2$. For the Elman ASRTN the number of cells is $C(h,m) = 2^h$, since $h \leq m = n + h$. Thus $C(1,3) = 2^1 = 2$. Therefore the number of open cells for both networks is $2^h - 0 = 2$ and the number of closed cells zero. The two open cells in the two-dimensional input space of the FFTN and the two in the three-dimensional input space of the ASRTN are illustrated in Figure 4.1. The type of a cell in the hidden space is determined by the type of its corresponding cell in the input space. The type of the cells in the hidden space of the FFTN and ASRTN is therefore open. Since the number of imaginary cells is $C_i(h,n) = 2^h - C(h,n)$ for the FFTN and $C_i(h,m) = 2^h - C(h,m)$ for the ASRTN, it is zero for both types of networks. This example shows that when $h \leq n$ and h context units are added to the input layer (as long as the number of inputs is not exceeded), there is no increase or decrease in the number of open, closed or imaginary cells.

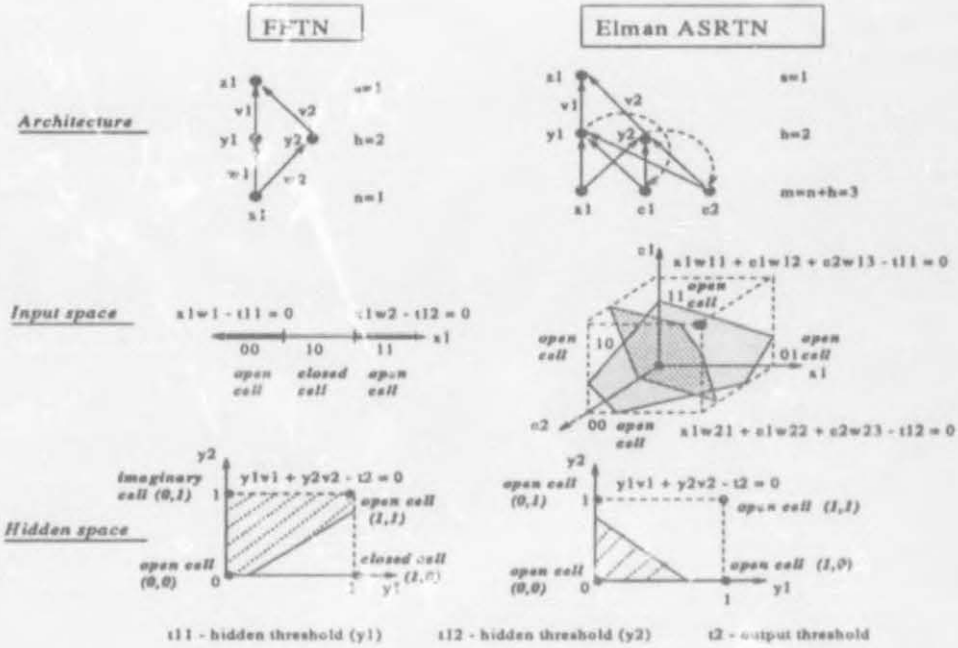


Figure 4.2: Example 2: Architecture, input and hidden space for *1:2:1* FFTN and Elman ASRTN.

Example 2: Let $n = 1$, $h = 2$, and $s = 1$. Figure 4.2 illustrates a *1:2:1* FFTN and the corresponding *1:2:1* Elman ASRTN in terms of network architecture, input and hidden space. In this example, each cell is labeled with a binary number y_1y_2 that specifies on which side of the hyperplanes (for the FFTN, y_1's hyperplane equation is $x_1w_1 - t_{11} = 0$ and y_2's equation is $x_1w2 - t_{12} = 0$) the points in that cell lie. The weights and thresholds of the FFTN are $w_1 = 0.7$, $w_2 = 0.6$, $t_{11} = 0.2$, $t_{12} = 0.5$, $v_1 = -0.7$, $v_2 = 0.8$, and $t_2 = -0.3$. The corresponding values

for the Elman ASRTN are $w_{11} = 0.5$, $w_{12} = 0.8$, $w_{13} = 0.4$, $t_{11} = 0.6$, $w_{21} = 0.5$, $w_{22} = 0.6$, $w_{23} = -0.8$, $t_{12} = -0.3$, $v_1 = 0.8$, $v_2 = 0.9$, and $t_2 = 0.6$. Since $h > n$, the number of cells for the *1:2:1* FFTN is $C(2,1) = \sum_{i=0}^{1} \frac{2!}{i!(2-i)!} = 3$. Although $h > n$, h is always smaller than $m = n + h$. Therefore the number of cells for the *1:2:1* Elman ASRTN is $C(1, 1+2) = 2^2 = 4$. The number of open cells for the FFTN is $C_o(2,1) = 2\sum_{i=0}^{0} \frac{(2-1)!}{i!(2-1-i)!} = 2$. For the ASRTN it is $2^h = 2^2 = 4$. The number of closed cells formed by the hyperplanes of the *1:2:1* FFTN is $C_c(2,1) = \frac{(2-1)!}{1!(2-1-1)!} = 1$. For the ASRTN the number of closed cells is zero. The number of imaginary cells for the FFTN is $C_i(2,1) = 2^2 - C(2,1) = 4 - 3 = 1$, while for the ASRTN it is $C_i(2,3) = 2^2 - C(2,3) = 4 - 4 = 0$. Two of the three cells in the one-dimensional input space of the FFTN are open (labeled 00 and 11 in Figure 4.2), while the other cell's input variables are bounded by the two hyperplanes (the closed cell is labeled 10). However, in the hidden space of the FFTN four different cells exist. The cell labeled 01 is one which is labeled by a hidden activation value that has no corresponding input values, thus being an imaginary cell. The function of this particular cell is to connect the two open cells in this case to form a disconnected decision region (consisting of the union of the cells labeled 00 and 11). Figure 4.2 illustrates the four open cells in the three-dimensional input space of the *1:2:1* ASRTN and the two-dimensional hidden space containing zero closed and imaginary cells and four open cells (as opposed to only two open cells for FFTN). This example shows that when $h > n$ and h context units are added to the input layer, the number of cells and open cells increases to 2^h, whereas the number of closed and imaginary cells decreases to zero.

In order to obtain a sense of numerical values involved, Table 4.1 enumerates the number of open, closed, and imaginary cells for *n:h:s* FFTNs and ASRTNs, with small values of h and n.

4.2.5 Interpretation of equations

When $h \leq n$, the number of cells for an *n:h:s* FFTN and its corresponding *n:h:s* Elman ASRTN is $2^h = \sum_{i=0}^{h} \frac{h!}{i!(h-i)!}$. In this case the h additional inputs (context units) of the ASRTN are not playing any role in the formation of cells, since the maximum number of cells has already been formed in the n-dimensional input space, namely 2^h. The FFTN does not have any closed or imaginary cells, so the addition of context units to the input layer cannot decrease the number of closed and imaginary cells any further.

However, when $h > n$, the context units have a very definite effect on the number and type of cells formed, and we can determine the effect accurately. Since the number of cells formed by the FFTN is $\sum_{i=0}^{n} \frac{h!}{i!(h-i)!}$, which equals the number of open cells plus the number of closed cells,

			FFTN				ASRTN			
n	h	m	$C(h,n)$	$C_o(h,n)$	$C_c(h,n)$	$C_i(h,n)$	$C(h,n)$	$C_o(h,n)$	$C_c(h,n)$	$C_i(h,n)$
1	1	2	2	2	0	0	2	2	0	0
2	1	3	2	2	0	0	2	2	0	0
3	1	4	2	2	0	0	2	2	0	0
1	2	3	3	2	1	1	4	4	0	0
1	3	4	4	2	2	4	8	8	0	0
2	2	4	4	4	0	0	4	4	0	0
2	3	5	7	6	1	1	8	8	0	0
3	2	5	4	4	0	0	4	4	0	0
3	3	6	8	8	0	0	8	8	0	0

Table 4.1: The number of open, closed, and imaginary cells for *n:h:s* FFTNs and ASRTNs, with small values of n and h.

the following equation holds

$$\begin{aligned} C(h,n) &= \sum_{i=0}^{n} \frac{h!}{i!(h-i)!} \\ &= 2\sum_{i=0}^{n-1} \frac{(h-1)!}{i!(h-1-i)!} + \frac{(h-1)!}{n!(h-1-n)!} \end{aligned} \tag{4.6}$$

The first term is the number of open cells (4.3), whereas the second term is the number of closed cells (4.4). The number of imaginary cells for a FFTN is equal to

$$\begin{aligned} C_i(h,n) &= 2^h - C(h,n) \\ &= \sum_{i=0}^{h} \frac{h!}{i!(h-i)!} - \sum_{i=0}^{n} \frac{h!}{i!(h-i)!} \\ &= \sum_{i=n+1}^{h} \frac{h!}{i!(h-i)!} \end{aligned} \tag{4.7}$$

The number of cells formed by a corresponding ASRTN can now be determined in terms of equations (4.6) and (4.7):

$$\begin{aligned} C(h,n+h) &= \sum_{i=0}^{h} \frac{h!}{i!(h-i)!} \\ &= \sum_{i=0}^{n} \frac{h!}{i!(h-i)!} + \sum_{i=n+1}^{h} \frac{h!}{i!(h-i)!} \\ &= 2\sum_{i=0}^{n-1} \frac{(h-1)!}{i!(h-1-i)!} + \frac{(h-1)!}{n!(h-1-n)!} + \sum_{i=n+1}^{h} \frac{h!}{i!(h-i)!} \end{aligned} \tag{4.8}$$

This equation gives an exact account of the effect of adding h context units to the input layer of a FFTN: the first term $2\sum_{i=0}^{n-1}\frac{(h-1)!}{i!(h-1-i)!}$ was the number of open cells in the FFTN case, which is now still open cells in the ASRTN case; the second term $\frac{(h-1)!}{n!(h-1-n)!}$ was the number of closed cells for the FFTN, which is now open cells for the ASRTN; the third term $\sum_{i=n+1}^{h}\frac{h!}{i!(h-i)!}$ was the number of imaginary cells for the FFTN, now also being open cells for the ASRTN. The FFTN's imaginary cells increase the number of open cells in the ASRTN's input space, which in turn leads to finer discrimination between data points. The addition of context units therefore extends the classification capabilities of a FFTN.

4.2.6 Classification Dynamics

In this section the focus is on the dynamics of the classification process in Elman ASRNNs. The dynamics is explored by analyzing the network's internal representation, i.e. the clusters formed by the hidden unit activation values of the network. We investigate problems that can be represented as finite state machines. The clusters formed by the hidden unit activations do not only represent a distributed representation of the network's internal states, but also represent the states of a finite state machine that is simulated by the Elman ASRNN. The network dynamics is depicted by the transitions between the states. To analyze the dynamics we have used four different applications, which vary in complexity. They are the more familiar temporal $\overline{XOR}$ problem, the detection of two consecutive ones, the detection of three consecutive ones, and the more complex addition task (presented in section 3.4.6). We start by constructing a transition diagram for each of the four applications to characterize the training data. For the addition task a FSM was also constructed for training data, which in turn enabled us to identify non-deterministic elements in the training data. As a spin-off experiment we investigated the training performance of different training strategies (see Chapter 3) for training with non-deterministic data versus training with deterministic data. The next step in identifying the simulated FSM, is to analyze the internal representation of the Elman ASRNN. To obtain a static representation of the network dynamics we use Hierarchical Cluster Analysis. If the ASRNNs are small enough, i.e. less than four input and context units for the input-context activation space, we can visualize the clusters formed in this space. The temporal $\overline{XOR}$ and two pattern detection applications only require networks that fall in the above-mentioned category, called *visualizable* networks. For these experiments, we use the same visualizable ASRNN, having one input and output, two context, and two hidden units. For this Elman ASRNN, which has $2^h = 4$ open cells in the input-context activation space, we visualize the clusters formed in these cells. However, for the addition task, which requires a more complicated ASRNN, we reduce the dimensionality by

using not only Principal Component Analysis, but also apply a fairly unfamiliar technique called Sammon Transformation Analysis [Sammon, 1969]. After identifying the simulated FSM from the clusters formed by these techniques, we determine the correspondence between the identified FSM and the one initially constructed for the training data.

4.2.6.1 Temporal $\overline{XOR}$

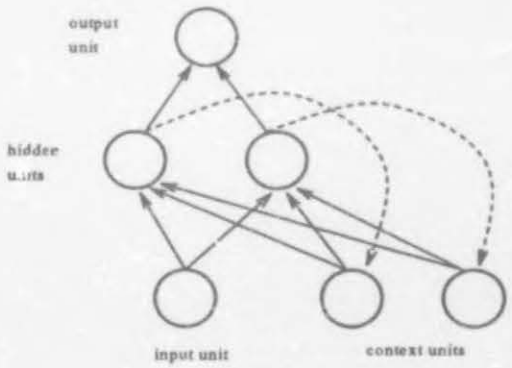


Figure 4.3: Elman ASRNN with one input, two context units, two hidden units, and one output.

In this section we analyze the classification dynamics of an Elman ASRNN with one input, two context, two hidden, and one output unit (see Figure 4.3) for the familiar temporal $\overline{XOR}$ ($notXOR$) application. We first construct a transition diagram to characterize the training data and to assist in identifying the FSM that is simulated by the network. The network is small enough so that we can visualize the 3-dimensional input-context space and 2-dimensional hidden activation space. Since we expect that the clusters formed by the hidden unit activations over time will indicate the states of the finite state machine of the network dynamics, we use Hierarchical Cluster Analysis to obtain these clusters. We then identify a Moore machine that corresponds to the internal representations of the network.

In the temporal $\overline{XOR}$ application the network must learn to detect two consecutive zeros or ones in a bit stream having only a one-bit input window (one input unit). The target value of the output unit is one when the previous and current inputs are identical, and zero otherwise. For example, if the input stream is 0001011 (where the rightmost bit is the current input), then the corresponding output stream is 0110001 (where the rightmost bit is the current output). Figure 4.4 shows the transition diagram constructed for the temporal $\overline{XOR}$ application to characterize the training input and output of the application. There are four basic transitions as indicated by A, D, B, and C, where the first two represent transitions with output one, and

Transition	Previous Input	Current Input	Output
A	0	0	1
B	0	1	0
C	1	0	0
D	1	1	1

Figure 4.4: Transition diagram for Temporal $\overline{XOR}$ application

the last two represent transitions with output zero.

(a) Visualization of Input-Context Space

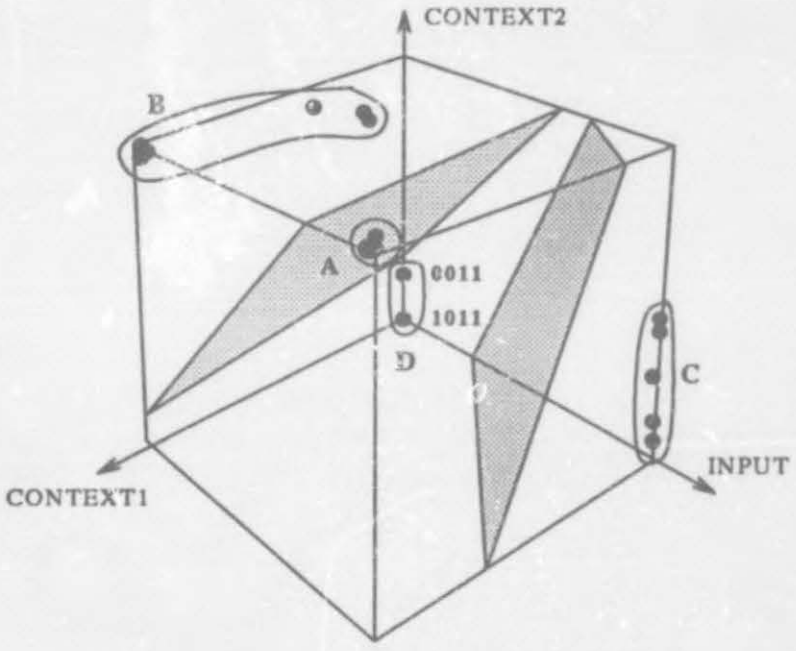


Figure 4.5: Visualization of input-context space of the Elman ASRNN for Temporal $\overline{XOR}$ classification

The small Elman ASRNN, which was trained with 100 single input patterns, found the task difficult to learn and reached an optimum of 74% accuracy on the test set (with no reset of context units). We then extracted the three input-context activation values (input and context

unit activations) as well as the two hidden unit activations over time, as the network processed the classification data, which consisted of 25 single input patterns. Since the context units operate on the same level as the input units, we have to visualize the space defined by the input and context unit activations, called the *input-context activation space*. Figure 4.5 illustrates a visualization of the classification data and the two hyperplanes (associated with the two hidden units) in the input-context activation space defined by INPUT (the input unit activation), CONTEXT1 and CONTEXT2 (the two context unit activations). To avoid visualization clutter, the hyperplanes are a threshold approximation of their original 3-dimensional sigmoid shape. Only three of the four open cells ($2^h = 4$) are visible, since the two hyperplanes intersect outside the figure. It is also apparent from the figure that both hyperplanes are needed to separate the four clusters correctly, causing it to be the smallest network to learn the task. The two clusters A and D, representing an output of one, are captured in the same open cell (in between the two hyperplanes) and located at opposite corners of the hypercube. Clusters B and C are located on the outside of the hyperplanes in different open cells, the former to the left of the left hyperplane and the latter to the right of the right hyperplane. Since there is no reset of the context unit activations during an epoch, all the previous input values influence the position of the current data points in the input-context space. For example, say we have an input history of three time steps, then the top data point in cluster D is labeled 0011 (where the rightmost bit is the current input) and the bottom data point labeled 1011. Both points have the same current and previous input (therefore being in the same cluster), but since their input two and three time steps back differ, they are situated at different locations in the input-context space. By visualizing the input-context activation space, four clusters were identified that correspond to the four main transitions of the Temporal $\overline{XOR}$ problem.

(b) Hierarchical Cluster Analysis

Hierarchical Cluster Analysis (HCA) is a method of finding the optimal partition of training vectors according to some similarity measure, such as Euclidean distance. The algorithm works by iteratively merging smaller clusters into larger ones. It starts with one data point per cluster, and then looks for the smallest distance between any two clusters. Those two clusters are merged so that they form a new cluster with the two earlier ones as subclusters, which gives rise to a branch in the cluster tree. The merging is repeated until one cluster is left. The results in the previous section were confirmed by a HCA of the hidden activation values as the network processed the 100 single patterns used for training. The hidden activation vectors along with their transition labels (which were determined by their corresponding inputs and outputs according to Figure 4.4), served as input to a cluster analysis program. The objective was to

determine whether the hidden activations are clustered according to the four main transitions of the transition diagram. In the graphical results of HCA, illustrated in Figure C.3 (Appendix C), each leaf in the tree corresponds to one of the four transitions. The figure reflects the network's performance on the test set, since there are a few misclassifications in each of the clusters. There are, for example, a few A's in the C cluster which indicate that the network struggles to remember the previous input when its current input is zero. This is also the case for cluster B, where a few D's illustrate the network's confusion with the previous input when its current input is one. A few A's appear in cluster D, indicating that the Elman ASRNN has learned the correct output, but confuses its input. Notwithstanding, the hidden activations are grouped according to the four main transitions.

(c) Identification of Moore Machine

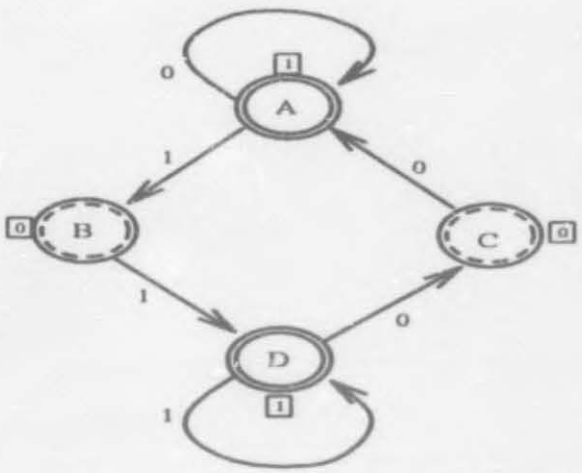


Figure 4.6: Moore machine of the Elman network dynamics for Temporal $\overline{XOR}$

In this section we identify the FSM that the network simulates. The visualization and HCA results indicate that the Elman ASRNN learns the four main transitions in Figure 4.4. The clusters obtained with these techniques correspond to the states of a Moore machine. A *Moore machine* is a quintuple (S, A, T, O, f), where S is a finite set whose elements are called states, A is a finite set called an input alphabet, $T : S \times A \rightarrow S$ is the transition function, O is the output alphabet, and $f : S \rightarrow O$ is the output function. Figure 4.6 presents the graphical representation of the Moore machine, which describes the hidden layer dynamics. The set of states is $S = \{A, B, C, D\}$, where B and C are the start states (denoted by the circles with a dash edge), and A and D the final states (denoted by the circles with two solid edges). The remaining task was to determine the Moore machine's transition and output functions (indicated in the figure by respectively the labeled arrows and labeled squares), as well as the input and

output alphabet, respectively $A = \{0, 1\}$ and $O = \{0, 1\}$. An Elman ASRNN can therefore learn to mimic closely a Moore machine for the temporal $\overline{XOR}$ application.

4.2.6.2 Detection of Two Consecutive Patterns

In this section we use an application for which the network must learn to detect two consecutive ones in a bit stream, to identify the FSM that is simulated by an Elman ASRNN with one input, two context, two hidden, and one output unit. The target value of the output unit is one when the previous and current inputs are one, and zero otherwise. This application, being less complex than temporal $\overline{XOR}$ (detection of not only consecutive ones, but also consecutive zeros), should provide for better simulation results than the latter, which in turn should lead to better visualization in terms of clearly grouped clusters. It also gives the opportunity to investigate the use of two hyperplanes (two hidden units) in a problem for which only one seems to be suffice. For temporal $\overline{XOR}$ two hyperplanes were necessary to separate the four clusters in the input-context space. Again we start by constructing a transition diagram for the training data, which is illustrated in Figure 4.7. There are four basic transitions as indicated by A, B, C, and D, where only the last one represents a transition with output one and the rest transitions with output zero. We proceed to visualize the input-context space and to use Hierarchical Cluster Analysis to obtain the clusters formed by the hidden activation values over time. Finally, we identify a Moore machine that corresponds to the Elman ASRNN's dynamics.

Transition	Previous Input	Current Input	Output
A	0	0	0
B	0	1	0
C	1	0	0
D	1	1	1

Figure 4.7: Transition diagram for the detection of two consecutive ones

(a) V' ualization of Input-Context Space

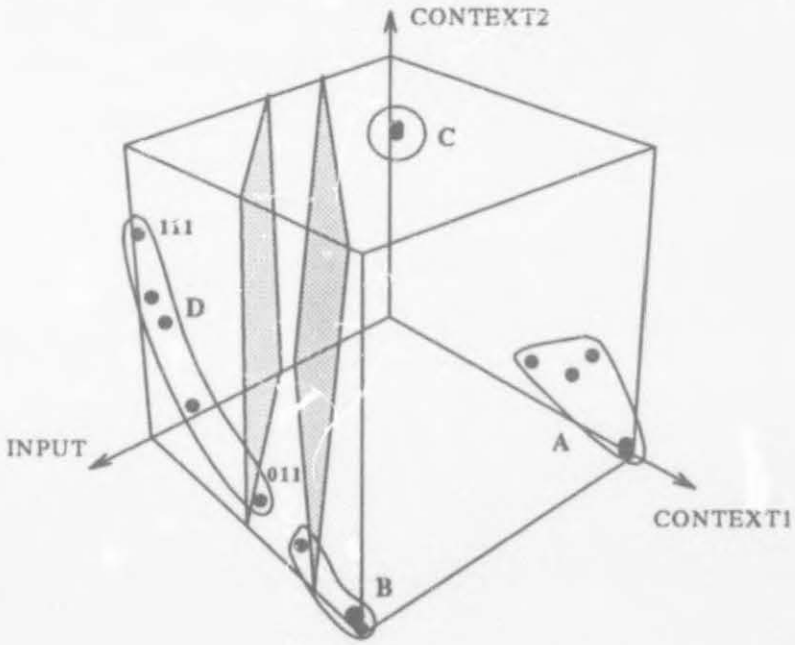


Figure 4.8: Visualization of input-context space of the Elman ASRNN for the detection of two consecutive ones

The Elman ASRNN was trained with 100 single input patterns and reached an optimum of 100% accuracy on the test set. Figure 4.8 illustrates a visualization of the classification data (25 single input patterns) and the two hyperplanes in the input-context space with a network performance of 94% accuracy on the classification set. From the figure it appears that one of the two hyperplanes is superfluous in separating the two groups of clusters. Thus only two open cells (of the four possible cells) are used, where clusters A, B, and C (representing an output of zero) are located in the one cell and cluster D (representing an output of one) in the other one. From this figure a misclassification can be visualized in the following manner: If the left hyperplane functions as the main separator, the rightmost data point of cluster D is misclassified; if the right hyperplane functions as the main separator, the leftmost data point of cluster B was a wrong classification. When the classification data for 100% accuracy was visualized, there were no data points located in between the two hyperplanes. The proximity of clusters B and C to the right hyperplane can be attributed to, respectively, a current and previous input of value one. Again, since there are no context unit activation resets during an epoch, all the previous input values influence the position of the current data points in the input-context space. For example, say we have an input history of two time steps, then the top data point in cluster D is labeled 111 (where the rightmost bit is the current input) and the data point to the right

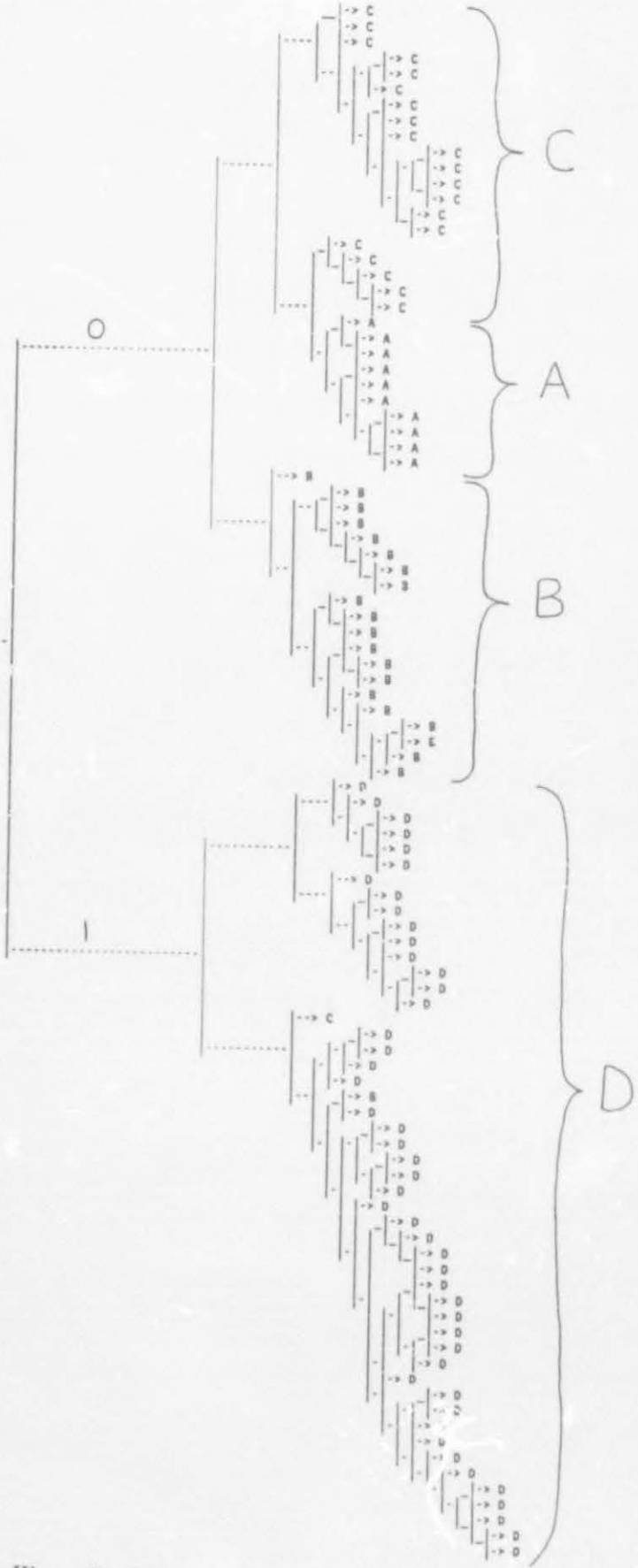


Figure 4.9: Hierarchical Cluster Analysis of the Elman ASRNN for the detection of two consecutive ones

of the left hyperplane, labeled 011. Both points have the same current and previous input, but since their input two time steps back differ, they are situated at different locations in the input-context space (although being in the same cluster).

(b) Hierarchical Cluster Analysis

The results in the previous section were confirmed by a Hierarchical Cluster Analysis of the 100 single patterns used for training. The graphical results of HCA, illustrated in Figure 4.9, reflect the network's performance of 100% accuracy on the test set, since the four clusters formed are close to being completely homogeneous. There are only two misclassifications in cluster D, and the couple of C's next to cluster A could lead to some confusion. From this figure it is evident that clusters representing an output of one are well separated from those representing a zero, and that the hidden activations are grouped according to the four main transitions.

(c) Identification of Moore Machine

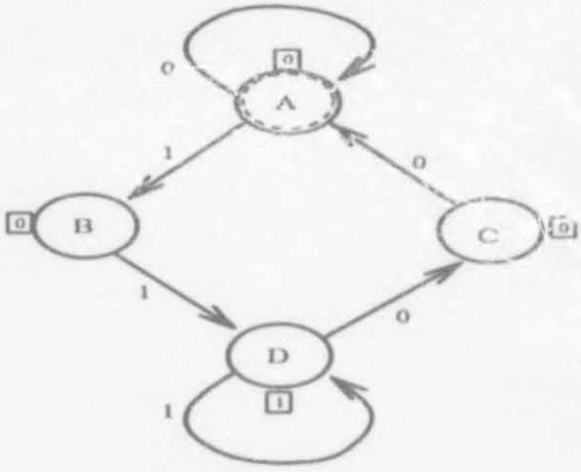


Figure 4.10: Moore machine of the Elman network dynamics for detecting two consecutive ones

From the graphs of the visualization of the input-context space and HCA, it follows that the Elman ASRNN learns the four main transitions in Figure 4.7. The clusters obtained correspond to the states of a Moore machine, illustrated in Figure 4.10. The set of states (represented by labeled circles) is $S = \{A, B, C, D\}$, where A is the start state (denoted by a circle with a dash edge) and all of them are final states. Again, the remaining task was to determine the Moore machine's transition and output functions (respectively indicated by labeled arrows and labeled squares), as well as input and output alphabet. Only the state labeled D has an output of one associated with it, whereas the others have an output of zero.

4.2.6.3 Detection of Three Consecutive Patterns

In this section we analyze the classification dynamics of the same Elman ASRNN as in the previous two sections, but this time using a more involved application for which the network must learn to detect three consecutive ones in a bit stream with only a one-bit input window (one input unit). The target value of the output unit is one when the input two time steps back, the previous input and the current input are all one, and zero otherwise. The complexity of this application is due to the fact that the network is supposed to remember input two time steps back. Again the network is small enough so that we can visualize the 3-dimensional input-context space and 2-dimensional hidden space (the reduction of dimensions is therefore not necessary). This enabled us to visualize the states of a more complicated finite state machine with eight possible transitions as opposed to the four of the previous two applications. Figure 4.11 shows the transition diagram constructed for this application. There are eight basic transitions as indicated by A through to H, where only the last one represents a transition with output one and the rest transitions with output zero.

Transition	Input (t-2)	Input (t-1)	Input (t)	Output (t)
A	0	0	0	0
B	0	0	1	0
C	0	1	0	0
D	0	1	1	0
E	1	0	0	0
F	1	0	1	0
G	1	1	0	0
H	1	1	1	1

Figure 4.11: Transition diagram for the detection of three consecutive ones

(a) Visualization of Input-Context Space

The Elman ASRNN was trained with 100 input patterns and reached an optimum of 100% accuracy on the test set. Figure 4.12 illustrates a visualization of the classification data (25 single input patterns) and the two hyperplanes in the input-context space with a network performance of 90% accuracy on the classification set. From the figure it appears that the right hyperplane functions as the main separator, separating cluster H correctly from the rest of the clusters.

Only two open cells are used, where cluster H (representing an output of one) appears in the one cell and the other clusters (representing an output of zero) in the other cell. Since the transitions B and F are closely grouped to each other, they form a natural cluster, labeled B-F. This is also the case for the A-E cluster. They give suitably a zero as output for a current input of respectively one and zero as well as for a previous input of zero (see Figure 4.11). The leftmost data point in cluster A-E was the starting point of this classification (and thus having no previous value), explaining why it is located so far apart from rest of the points in the cluster. The distance between the data points in cluster H can be attributed to the 90% accuracy as well as their input history three time steps back, being 1111 and 0111 for respectively the leftmost and the rightmost data point. The rest of the clusters are clearly grouped. Since cluster D contains data points labeled 011 (where the rightmost bit is the current input) and cluster B-F contains points labeled 001 and 101, their proximity to the right separating hyperplane indicates them acting as a thoroughfare in the path from the rightmost clusters, having a current input of zero, to cluster H, representing three consecutive ones as input. Consider, for example, the path of migration pictured in Figure 4.12. It starts in cluster A-E with data point 000, proceeds to cluster B-F with point 001, then moves to 011 in cluster D, and finally ends with data point 111 in cluster H.

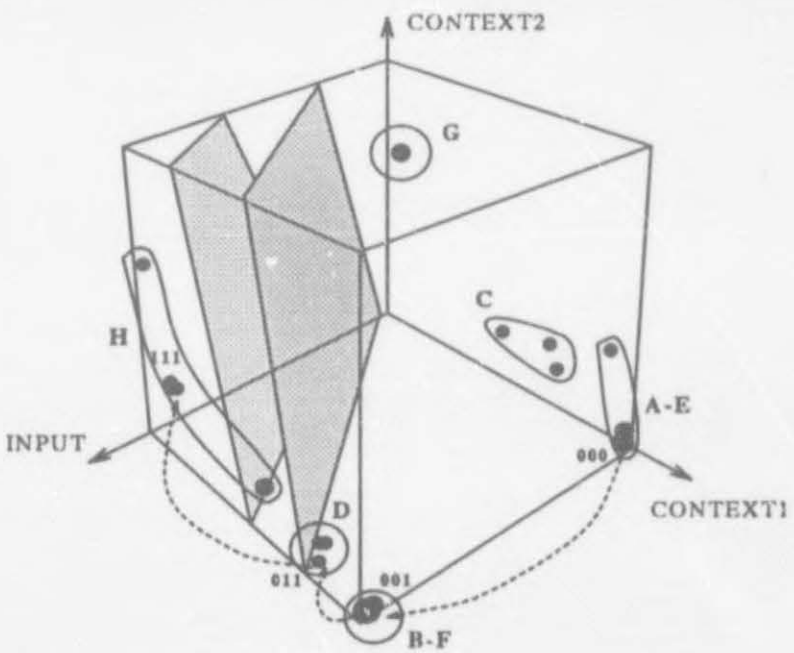


Figure 4.12: Visualization of input-context space of the Elman ASRNN for the detection of three consecutive ones

(b) Hierarchical Cluster Analysis

Again the results in the previous section were confirmed by a Hierarchical Cluster Analysis of the 100 single patterns used for training. The graphical results of HCA, illustrated in Figure C.4 (in Appendix C), reflect the network's performance on the test set, since there are only a few misclassifications in the six clusters formed. Since there are only two transitions of type A in the training data, it suggests why the network has difficulty with classifying them and why they are located at different positions in the tree. From the leftmost branching in the graph it is evident that clusters representing an output of one are well separated from those representing a zero, and the rest of the branching indicates that the hidden activations are grouped according to the six main transitions.

(c) Identification of Moore Machine

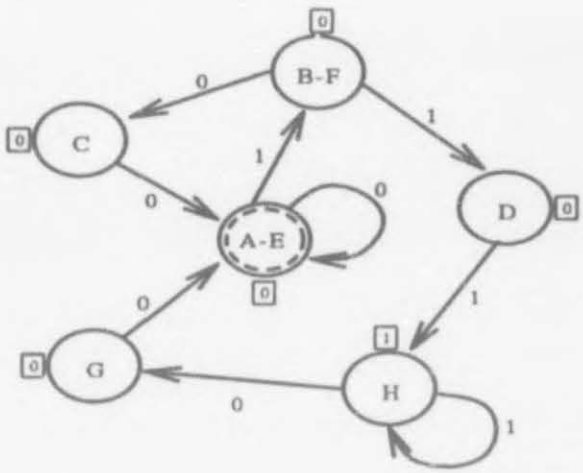


Figure 4.13: Moore machine of the Elman ASRNN dynamics for the detection of three consecutive ones

The visualization graphs in Figure 4.12 together with the graphical results of the HCA, show clearly how the hidden activations classify six main transitions. The clusters obtained identify the Moore machine's states (denoted by labeled circles in Figure 4.13), S = {A-E,B-F,C,D,G,H}, where A-E is the start state (denoted by a circle with a dash edge). The transition and output functions are represented respectively by labeled arrows and labeled squares. An Elman ASRNN can therefore learn to simulate a Moore machine for the detection of three consecutive patterns, both in its behaviour (the same input and output) and in its state representation (the six identified states).

4.2.6.4 Applications using 2^h open cells

In this section we analyze the input-context activation space of an Elman ASRNN (see Figure 4.3) for an application that can be learned by using all four open cells and one that needs five open cells. For the first application the network must learn to associate four different output values to the possible combinations of previous and current input values in a bit stream. The target output values appear in the transition diagram shown in Figure 4.14. For this application the network reached an optimum of 98% accuracy on the test set. In the visualization of the input-context space - illustrated in Figure 4.14 - each of the classes A, B, C, and D appear in a separate cell, thereby occupying the $2^h = 4$ possible open cells. The next application illustrates a case where the four open cells of the Elman network are not enough to correctly learn the task at hand.

Class	*Input t-1*	*Input t*	*Output t*
A	0	0	0.10
B	0	1	0.35
C	1	0	0.60
D	1	1	0.85

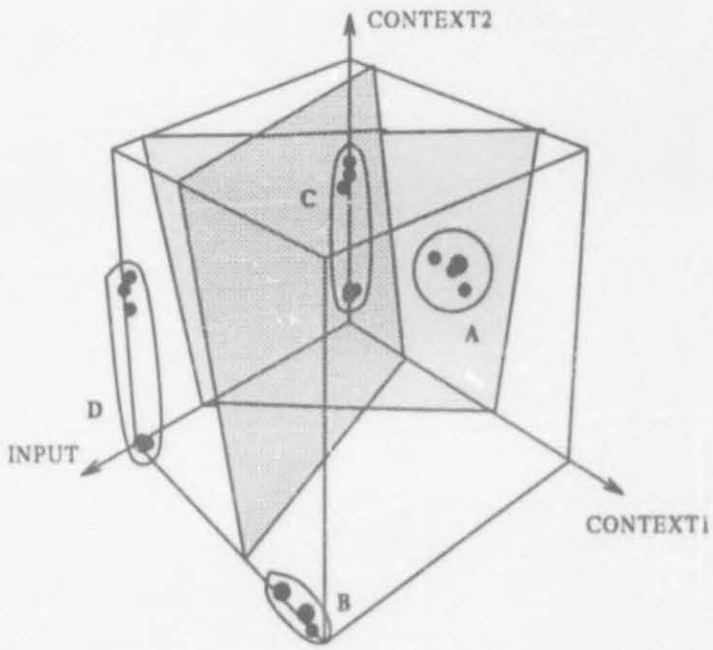


Figure 4.14: Visualization of input-context space of the Elman ASRNN for an application that uses all four open cells

For the second application the network must learn to associate five different output values to each of the possible combinations of input values zero, one and two time steps back in a bit stream (see the transition diagram in Figure 4.15). Transitions B and C belong to the same class, since both have the same target output value. This is also the case for transitions D and E, as well as F and G. Transitions A and H each define their own class. The Elman ASRNN with one input unit, two hidden and two context units, and one output unit was not able to

correctly learn the task, since the accuracy on the test set fluctuated between 42% and 66%. In the visualization of the input-context space (illustrated in Figure 4.15) all four cells are occupied by clusters belonging to the same class, except the cell containing clusters H, F and G. This grouping of clusters would only be correct if the target output value of transition H is changed to that of transitions F and G. However, to learn the task correctly, one extra cell is needed for cluster H. This could only be achieved by adding another hidden unit, which would in turn increase the number of open cells to $2^3 = 8$.

Class		Input t-2	Input t-1	Input t	Output t
1	A	0	0	0	0.00
2	B	0	0	1	0.25
	C	0	1	0	0.25
3	D	0	1	1	0.50
	E	1	0	0	0.50
4	F	1	0	1	0.75
	G	1	1	0	0.75
5	H	1	1	1	1.00

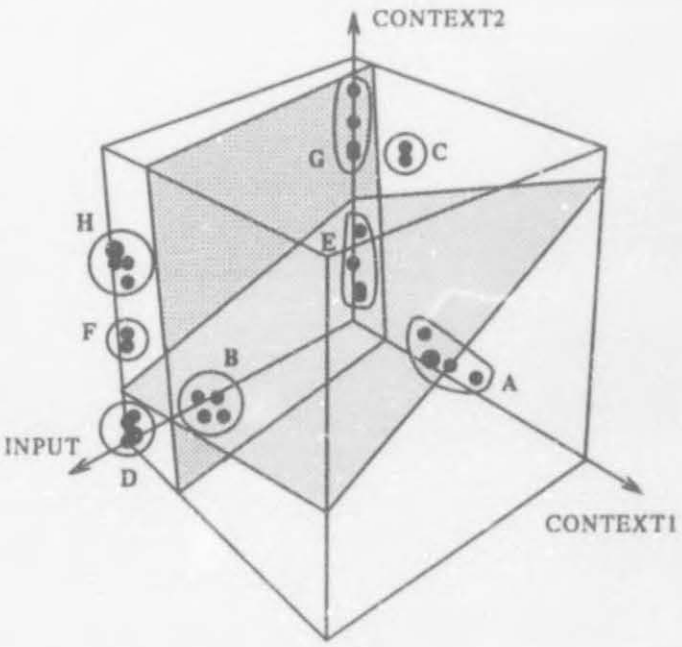


Figure 4.15: Visualization of input-context space of the Elman ASRNN for an application that needs five open cells

4.2.6.5 ADDITION

This section presents the classification dynamics analysis of the Elman ASRNN for the addition task [Ludik *et al*, 1994]. For the addition task (see section 3.4.6), which is more complex than the previous ones, an Elman ASRNN with five input, 16 context, 16 hidden, and six output units is used. To visualize the network dynamics, we reduce the dimensionality of the hidden activation space (in contrary to the previous tasks) by using two different techniques, Principal Component Analysis and Sammon Transformation Analysis. We again use Hierarchical Cluster Analysis to obtain a static representation of the network dynamics and then identify a Moore machine representing the internal representations of the network.

(a) Construction of Mealy Machine

We constructed a Mealy machine to characterize the addition task more precisely and to aid in identifying the finite state machine of the network dynamics. The Mealy machine in Figure 4.16 describes all the input-output combinations of the four possible actions in the addition problem. The actions are to write the *result*, to remember the *carry*, shift to the *next* column of digits, and indicate if *done* (see Figure 3.7). A *Mealy machine* is a 5-tuple (S, A, T, O, f), where S is a finite set of states, A is a finite set called an input alphabet, $T : S \times A \rightarrow S$ is the transition function, O is the output alphabet, and $f : S \times A \rightarrow O$ is the output function. For the Elman ASRNN of this problem the input patterns are the input alphabet of the Mealy machine, whilst the target output patterns are the output alphabet. Each transition represents a specific group of input-output transitions, which is specified in Tables 4.2 and 4.3. The top half of the Mealy machine describes the input-output combinations involved in zero or one carry, whereas the bottom half depicts those input-output transitions involved in more than one carry (top and bottom halves indicated in the figure).

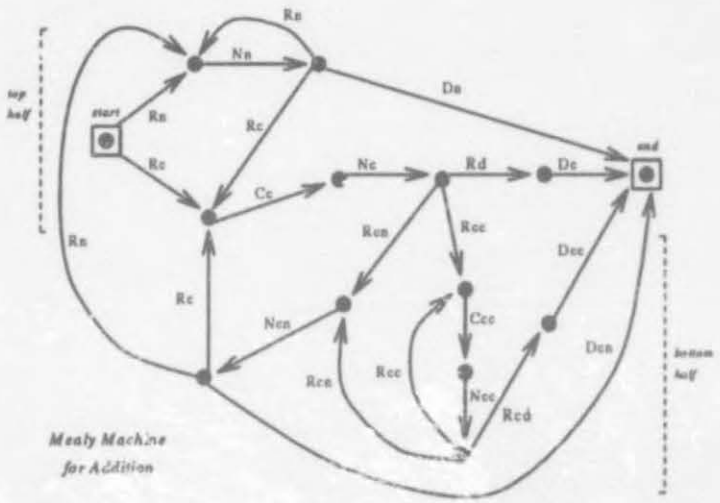


Figure 4.16: Mealy Machine for Addition

The *result* input-output combinations are denoted by Rx, where x is the type of result action indicated by N, C, D, CN, CC, and CD. R_N represent the result actions that lead to *next* actions, whereas R_C actions lead to *carry* actions. R_{CN} and R_{CC} are result actions, which incorporate the changes in the result field due to carry actions earlier in the current temporal pattern. They represent result actions that respectively lead to next and carry actions. R_D and R_{CD} are the final result actions that lead to *done* actions, where the former is part of a temporal pattern that only includes one carry, while the latter's temporal pattern includes more than one

carry.

The *carry* input-output combinations are denoted by Cx, where x is the type of carry action indicated by C and CC. C_C represent the first carry actions in a temporal pattern, while C_{CC} indicates the successive carry actions.

The *next* input-output combinations are denoted by Nx, where x is the type of next action indicated by N, C, CN, and CC. N_N represent next actions contained in a temporal pattern with no carry actions earlier in the temporal pattern, whereas N_C actions indicate one carry action earlier in the temporal pattern. N_{CN} and N_{CC} are next actions which indicate more than one carry action earlier in the temporal pattern. They differ in that the latter's preceding action is a carry (C_{CC}), whilst in the former's case it is a result action (R_{CN}).

The *done* input-output combinations are denoted by Dx, where x is the type of done action indicated by N, C, CN, and CC. D_N represent done actions that are preceded by a next action (N_N), whereas D_C actions are preceded by a result action (R_D) which is due to a carry action. D_{CC} represent done actions which are performed after more than one carry and preceded by a result action (R_{CD}). D_{CN} represent done actions which are performed after at least one carry and preceded by a next action (N_{CN}).

Input	Output							
	R_N	R_C	N_N	C_C	N_C	D_N	R_D	D_C
0000	100000		001000			000100		
0001	100010		001000			000100		
0010	100010		001000			000100		
0011		100000		010000	001000		100001	000100
0100	100001		001000			000100		
0101	100011		001000			000100		
0110	100011		001000			000100		
0111		100001*		010000	001000		100001*	000100
1000	100001		001000			000100		
1001	100011		001000			000100		
1010	100011		001000			000100		
1011		100001*		010000	001000		100001*	000100
1100	100010		001000			000100		
1101		100000		010000	001000		100001	000100
1110		100000		010000	001000		100001	000100
1111		100010		010000	001000		100001	000100

Table 4.2: Mealy machine transitions for zero or one carry. The * indicates a non-deterministic transition.

Table 4.2 specifies the Mealy machine transitions for temporal patterns that include zero or one carry action, whereas the temporal patterns of Table 4.3 include more than one carry. In both tables only the one column of digits (the top and bottom digits) are shown as input. The *end-of-input* bit of the input is not shown, because it is zero for all actions except for the done actions (D_N, D_C, D_{CN} and D_{CC}) when its value is one. The other four input bits indicate the bottom and top digit (two bits each). The six output bits indicate the four actions (one bit each) and the lower order result of the sum (two bits). For illustration purposes the example in Figure B.3 (Appendix B) is used. On the first time step the input is 1111 (representing 11 as the bottom and 11 as the top digit) and the corresponding output is 100010 (representing the write of the lower order result 10), which correlates to the R_C action in Table 4.2. Since the sum produced a carry, the action on the second time step is the C_C carry action (010000) for the same input 1111. On time step three, the shift to the next column of digits is indicated by the N_C next action (001000). On the fourth time step, the sum of the new bottom and top digits, respectively 01 and 00, plus the carry of the previous column is 01+00+01=10. Since the sum did not produce a carry, the corresponding output action for the 0100 input is therefore $R_C N$ (100010), indicating a result action that leads to a next action and incorporates the changes in the result field due to a carry action earlier in the temporal pattern (see Table 4.3). The three-column addition is completed with a D_{CC} done action for which the *end-of-input* bit is one.

(b) Training: Non-determinism versus Determinism

All the Mealy machine transitions in Tables 4.2 and 4.3 are deterministic, except those marked with an asterisk. In Table 4.2 there is a non-deterministic choice (the next state cannot be determined from the current state and current input) between the result actions R_C and R_D when the input is 0111 and 1011, i.e. identical output patterns corresponding to different result actions exist for a specific input. In Table 4.3 the non-deterministic choice is between the result actions R_{CC} and R_{CD} when the input is 0011, 1101, and 1110. These non-deterministic transitions were not evident from the training data and were only detected after the construction of the Mealy machine. Since it is easier to learn deterministic transitions than non-deterministic ones, we investigated ways of eliminating non-deterministic transitions from the training data without effecting the top and bottom digits and the corresponding output sequences involved in the addition task. One way to make these choices deterministic is to change the *end-of-input* bit into a one for R_D and R_{CD} in order to distinguish them uniquely from respectively R_C and R_{CC}. Thus every output pattern corresponding to an action is uniquely mapped onto a specific input pattern. This is also logically plausible, since R_D and R_{CD} are the only result actions

Input	Output							
	R_{CN}	R_{CC}	N_{CN}	C_{CC}	N_{CC}	D_{CN}	R_{CD}	D_{CC}
0000	100001		001000			000100		
0001	100011		001000			000100		
0010	100011		001000			000100		
0011		100001*		010000	001000		100001*	000100
0100	100010		001000			000100		
0101		100000		010000	001000		100001	000100
0110		100000		010000	001000		100001	000100
0111		100010		010000	001000		100001	000100
1000	100010		001000			000100		
1001		100000		010000	001000		100001	000100
1010		100000		010000	001000		100001	000100
1011		100010		010000	001000		100001	000100
1100	100011		001000			000100		
1101		100001*		010000	001000		100001*	000100
1110		100001*		010000	001000		100001*	000100
1111		100011		010000	001000		100001	000100

Table 4.3: Mealy machine transitions for more than one carry. The * indicates a non-deterministic transition.

leading to done actions. The next interesting step was to determine the difference in training performance when training with non-deterministic data (not an unique input-output mapping) versus deterministic data (an unique input-output mapping).

Training Strategies	Average # updates	Std. D. # updates	Average $\sum E_A$	Std. D. $\sum E_A$	Best # updates	% improvement comp. to FST
IST	20858	6082	1059.95	256.38	10995	18.5%
IICT	21470	3104	1130.77	123.22	17194	16.1%
ICT	22274	7043	1085.61	171.91	15641	12.9%
CST	23337	7523	1090.65	171.89	16364	8.3%
FST	25586	10071	1134.36	134.09	16135	-

Table 4.4: A comparison of the average *addition* simulation results for the deterministic case

The training performance of the different training strategies, FST, CST, ICT, IICT, and IST were investigated for these two cases. For each training strategy, ten simulations were performed with different initial sets of weights. Again the average and standard deviation of the number of updates, as well as the average and standard deviation of the sum of the Euclidean distance of all the weight changes were determined. The summary of the simulation results for the deterministic case appears in Table 4.4 (percentage improvement is compared to FST), whereas the simulation

results for the non-deterministic case were summarized in Table 3.11 (section 3.4.1). The graph in Figure 4.17 highlights the results of a particular simulation for the deterministic case, where IST outperformed all the other training strategies with a 47% improvement when compared with FST. The corresponding graph for the non-deterministic case can be found in Figure 3.16 (section 3.4.1), where all four training strategies improved the number of updates by more than 40% compared to FST, IST being the pick of the strategies by achieving 53.3%.

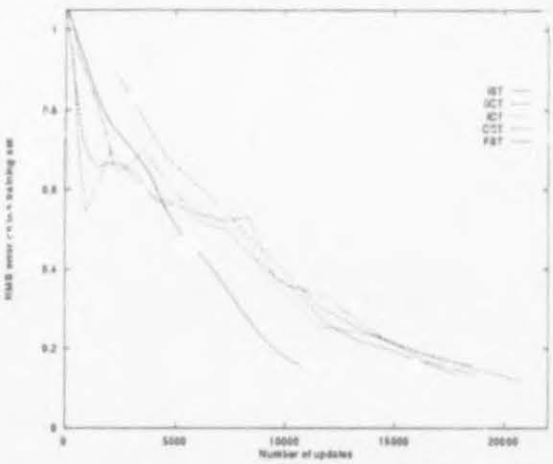


Figure 4.17: The performance of the Elman ASRNN for Addition in the deterministic case

There is a substantial difference in training performance when training with non-deterministic data versus training with deterministic data. This is confirmed by the results in the last column of Table 4.5, where all the training strategies performed much better with the deterministic data. All of them achieved noteworthy improvements of between 34% and 44%. We suspected that training would be easier with the deterministic data, but were quite surprised at the vast improvements. Only one bit in 149 input patterns was changed out of a possible 2305 input patterns with a length of 11 bits (that is only about 0.6% change in the total fixed training set). These results show how the finite state machine of the training data can be useful in eliminating possible non-deterministic elements.

In the following sections we analyze the hidden unit activations by using Sammon Transformation Analysis, Principal Component Analysis, and Hierarchical Cluster Analysis, how these techniques identify the states of a Moore machine for the network dynamics, and how it compares to the previous Mealy machine constructed by hand.

For purposes of analysis we have used the weight matrices of the best training strategy, IST, in the classification process of 8-10 column addition. We have extracted the 16 hidden unit activations over time, as the Elman ASRNN processed the classification data, which consisted

Training Strategies	Non-deterministic		Deterministic		Improvement of Determinism vs Non-determinism
	Updates	Improvement compared to FST	Updates	Improvement compared to FST	
IST	34700	24.4%	20858	18.5%	40.0%
IICT	32703	28.7%	21470	16.1%	34.3%
ICT	36186	21.1%	18450	12.9%	38.4%
CST	39478	13.9%	18760	8.8%	40.9%
FST	45870	-	20745	-	44.2%

Table 4.5: A comparison of the average *addition* simulation results for the non-deterministic versus deterministic case

of ten temporal patterns constituting 233 single input patterns.

(c) Sammon Transformation Analysis

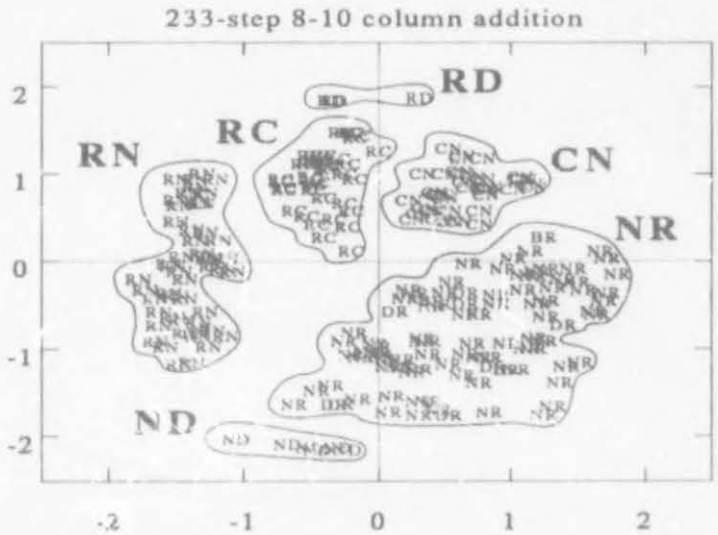


Figure 4.18: Sammon Transformation Analysis of 233-step 8-10 column addition

Sammon Transformation Analysis (STA) [Sammon, 1969] is a data transformation technique that maps multidimensional vectors onto two or three dimensional vectors, whose intervector distances tend to approximate those of the multidimensional vectors. The input data set consists of N L-dimensional vectors $\vec{x}_i$, $i = 1, 2, \ldots, N$. The output vector set consists of N M-dimensional ($M = 2$ or 3) vectors $\vec{y}_i$, $i = 1, 2, \ldots, N$. The distance between vectors $\vec{x}_i$ and $\vec{x}_j$ is defined as d^*_{ij}. Similarly, the distance between $\vec{y}_i$ and $\vec{y}_j$ is defined as d_{ij}. Choose a random initial configuration for the M-space vectors and compute the distances d_{ij} by using a distance

measure such as the Euclidean distance. The distances are used to define an error value E:

$$E = \frac{1}{\sum_{i<j}^{N} d^*_{ij}} \sum_{i<j}^{N} \frac{(d^*_{ij} - d_{ij})^2}{d^*_{ij}}$$

The vectors $\vec{y}_i$ are now adjusted, using a steepest descent algorithm, so as to minimize the error E. The final result is a set of vectors $\vec{y}_i$, $i = 1, 2 \ldots, N$ whose intervector distances tend to approximate the intervector distances of the vectors $\vec{x}_i$, $i = 1, 2, \ldots, N$. For our experiments the input and output dimensions were respectively $L = 16$ and $M = 2$.

Figure 4.19: Sammon Transformation Analysis of 233-step 8-10 column addition – Mealy machine correspondence

In Figure 4.18 we show the projection of the hidden units vectors onto two dimensions as the network is doing the 233-step addition. The clusters formed by the projected hidden unit activations correspond vividly to the different types of actions that the network are required to learn. Six clusters can be identified that correspond to the main transitions of the four different actions, namely *Next-Result(NR), Result-Next (RN), Carry-Next (CN), Result-Carry (RC), Result-Done (RD),* and *Next-Done (ND)*. The above-mentioned transition labels, each one associated with its corresponding hidden unit activation vector, formed natural clusters according to the transitions between the different actions. The separation between clusters was made more observable by indicating the edges of the clusters, which were determined by the outermost transition labels in each clusters. Along the x-axis the network is distinguishing between a *Next* that follows a *Carry (CN)* versus one that follows a *Result* action *(RN)*. Along the y-axis the network is distinguishing between a *Done* that follows a *Result (RD)* versus one that follows a *Next* action *(ND)*.

Figure 4.19 illustrates the correspondence between the STA projected hidden activation vectors and the Mealy machine transitions in the previous section. The objective was to determine if the hidden activations are clustered according to the transitions of the Mealy machine. The following mapping exists between the transition clusters in Figure 4.18 and the Mealy machine transitions: $NR = \{R_N, R_C, R_D, R_{CN}, R_{CC}, R_{CD}\}$; $RN = \{N_N, N_{CN}\}$; $CN = \{N_C, N_{CC}\}$; $RC = \{C_C, C_{CC}\}$; $RD = \{D_C, D_{CC}\}$; and $ND = \{D_N, D_{CN}\}$. Another interesting result is the distinct separation between clusters that represent actions involved in a carry (located in the top right half of Figure 4.19) and clusters representing other actions (located in the left bottom half). The figure also shows the existence of two groups of actions in the NR cluster, namely a no-carry group $\{R_N, R_{CN}\}$ and a carry-group $\{R_C, R_D, R_{CC}, R_{CD}\}$. This was not evident in Figure 4.18 where the more general transition label NR was used to denote transitions from next actions to result actions without distinguishing between different types of result actions (which are provided by the Mealy machine labels).

(d) Principal Component Analysis

Principal Component Analysis (PCA) is a technique for mapping multidimensional vectors onto a new set of orthogonal linear vectors, where the first principal component is such that the projections of the given points onto it have maximum variance among all possible linear coordinates; the second principal component has maximum variance subject to being orthogonal to the first; and so on. In Figure 4.20 we show the projection of the hidden units vectors onto the plane of the first two principal components as the network is doing the 233-step addition. The figure illustrates the correspondence between the PCA data and the Mealy machine transitions, which is similar to the STA correspondence. The *Result* actions are generally in the left half of the space, wheras the *Nexts* and *Carrys* are in the right half. Along the second principal component the network is distinguishing between a *Next* that follows a *Carry (CN)* versus one that follows a *Result* action. Clusters that represent actions involved in a *Carry* (located in the top half of the space) can be *linearly* separated from clusters representing actions not involved (located in the bottom half).

A graph similar to Figure 4.18 was also generated to show the PCA projection of the hidden units vectors onto two dimensions as the network is doing the 233-step addition (see Figure C.1 in Appendix C). Again six clusters were identified that correspond to the main transitions of the four actions. We have also obtained similar results by plotting the first principal component at time t versus $t+1$, which essentially gives a mapping from the context vector to the next hidden vector.

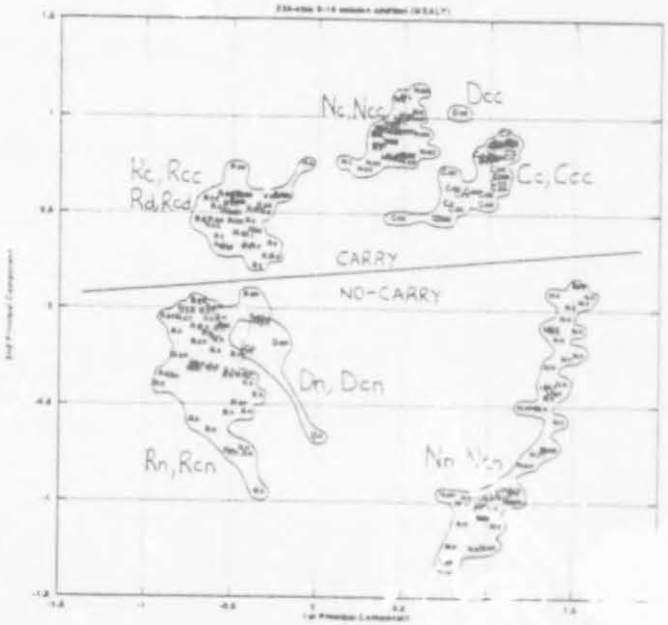


Figure 4.20: Principal Component Analysis of 233-stack 8-bit column addition – Mealy machine correspondence

By comparing Figures 4.19 and 4.20, it is evident that STA produces superior clustering results as opposed to PCA for this experiment, since some of the clusters in Figure 4.20 are not well separated from each other (for example clusters $\{R_N, R_{CN}\}$ and $\{D_N, D_{CN}\}$). We conjecture that this will be the case for other experiments as well, since STA attempts to preserve the intervector distances, whereas PCA discards them.

(e) Hierarchical Cluster Analysis

The hidden activation vectors along with their corresponding transition labels (*RD*, *RC*, *CN*, *NR*, *RN*, and *ND*) served as input to a Hierarchical Cluster Analysis program. The optimal partition of the hidden activation vectors was obtained according to the Euclidean distances between each pair of these vectors. Figure 4.21 shows the graphical results of this analysis, where each leaf in the tree corresponds to a particular transition from one action to another. This figure shows how the activation patterns are grouped according to the six main transitions between the different actions, as was the case with STA (Figure 4.18) and PCA (Figure C.1). We have also plotted a graph illustrating the correspondence between the HCA data and the Mealy machine transitions (see Figure C.2), which is similar to the STA and PCA correspondences.

(f) Identification of Moore Machine

The STA, PCA, and HCA clustering analysis techniques show that the hidden activations over time can be grouped into six clusters that correspond to the main transitions of the result, next,

Figure 4.21: Hierarchical Cluster Analysis of 233-step 8-10 column addition

carry and done actions. The clusters obtained with these techniques correspond to the states of a Moore machine, which is graphically presented in Figure 4.22. The six clusters are the states $S = \{NR, RC, CN, RN, ND, RD\}$ (the states are denoted by labeled circles), where NR is the start state (indicated by a circle with a dash edge) and RD and ND the final states (indicated by circles with two solid edges). The Moore machine's transitions are represented by labeled arrows and the output associated with each state is represented by a labeled square. The input symbols of the input alphabet $A = \{0c, 0n, 1c, 1n\}$ are represented in such a manner that 0 or 1 indicates respectively *not-end-of-input* and *end-of-input*, and c and n respectively *carry* and *no-carry*. The output alphabet is defined by $O = \{R, C, N, D\}$, where the output symbols respectively are *Result*, *Carry*, *Next*, and *Done*. Each state of the Moore machine corresponds to Mealy machine transitions, as described in section 4.1.4.3.

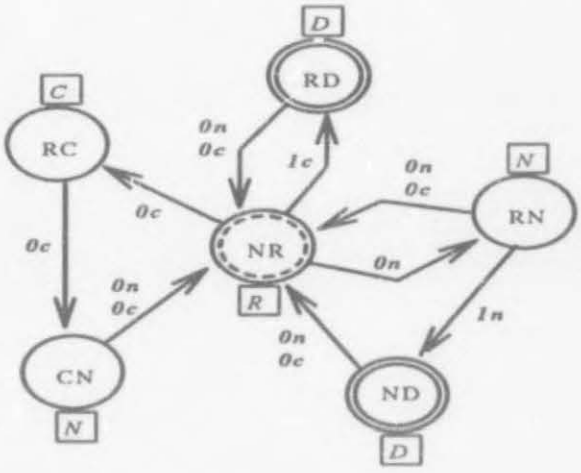


Figure 4.22: Moore machine of the Elman network dynamics for Addition

4.2.7 Learning Dynamics

In this section we analyze the dynamics of the learning process in Elman ASRNNs by first discussing the important role of the context units during learning. We proceed by visualizing the evolution of clusters formed by Sammon Transformation Analysis during the learning process for the addition task and map the stages in cluster formation onto points on the learning curve. The goal is to observe how the network learns to distinguish between different actions and how clusters are formed. The visualizion of how the hidden activations evolve over time for particular input of temporal $\overline{XOR}$, enable us to study temporal paths formed in the hidden activation space during learning. We expect to observe the ASRNN's learning behaviour (learning difficulties in particular) from the initial position in the hidden space (determined by the initial random

weights) to the final position (in one of the clusters of temporal $\overline{XOR}$ which have already been identified in section 4.2.6.1). We then examine the Elman ASRNN with different temporal window sizes, using the application for detecting two consecutive ones, to determine how the reset of the context unit values affect the network's performance.

4.2.7.1 Role of the context units

As we have mentioned before, the input values to the context units are provided by the network itself, specifically by the values of the previous hidden units and fixed one-to-one connections. These special context inputs, apart from the normal input units of the input layer, give the Elman ASRNN the capability to store temporal patterns or predict the next input. Since the context units together with the input units activate the hidden units, the context units can be considered to be part of the input to the ASRNN.

As long as the Elman ASRNN learns (changes its weights), the input (consisting of the input and context units) continually changes. The latter is valid, since changing the weights causes different hidden activations to be generated, which implies continually changing context unit values. Thus, although the input unit values do not change from epoch to epoch, the context unit values do change. In the classification phase the input-context vectors are constant for every epoch, since the weights do not change. However, for the learning phase they continue to vary until the weight changes stabilize. This has the effect that input-context data points continue to migrate along with the hyperplanes in the input-context space on every time step. This network behaviour does not correspond to the behaviour of the feedforward network in the input space, since for the latter the same inputs are presented to the network in every epoch and only the hyperplanes move as the weights change. It is interesting to note that the migration of data points along with hyperplanes do occur in the hidden spaces of both ASRNN and feedforward networks during learning, since the hidden activations constantly change as the weights change. Therefore the migrating behaviour takes place on both input and hidden levels of the Elman ASRNN.

From the above-mentioned perspective, one also gains some insight in the role of the context units in the weight updating scheme, whether weights are updated after every input pattern (incremental updating) or after every epoch of input patterns (epoch updating). For incremental updating, the context values on each time step will be different from the context values generated by epoch updating, since for the latter the weight changes only take place at the end of the epoch and its effect on the context units will only be visible in the following epoch.

We have also investigated the initialization of context units and determined that the particular initialization value only has an effect if the context units are frequently being reset during training. If no context reset takes place, the initialization of context units is unimportant and does not influence the network's behaviour. Similar performance results were achieved when different initial context values were used in the addition application with FST learning. For example, when the initial context values were respectively set to 0 and 0.5, both ASRNNs reached the desired RMS criterion of 0.15 after 21 epochs, with their respective RMS values being 0.147 and 0.146. Since the context values are not reset for addition and only a done bit is used to specify the end of the temporal pattern, the initialization value of the context units does not have any effect on the network's behaviour during the learning phase. We expand further on the issue of context reset in section 4.2.7.4.

Since the input values to the context units are viewed as part of the network's input, it can also be considered as part of the ASRNN's training set. This perspective influences the way in which the training sets of an ASRNN and a standard feedforward network are being viewed. The training set of a feedforward network (consisting of values for the input and output units) does not change and is repeated at every epoch. However, the "training set" of an ASRNN (consisting of values for the input, context and output units), changes continually, since the values for the context units change as the network learns. As learning proceeds, its "training set" constantly improves as the network comes closer to a solution, since the values of the context units come closer to their desired values (in the final clusters). The number of "training examples" increases until the weight changes stabilize, because the context unit values differ from one weight change to another.

4.2.7.2 Evolution of SAMMON Cluster Formation

In this section we explore the learning dynamics of the Elman ASRNN for addition by visualizing how the internal representations evolve over time during training. We observe how the network learns to distinguish between different actions and how clusters are formed. For this exploration we have selected the best training strategy for addition, namely IST, and determined ten inspection points, one after ever fourth incremental subset. Figure 4.23 shows these inspection points plotted on the RMS learning curve of IST, while Table 4.6 gives the simulation results at each inspection point. We then map each inspection point on the learning curve to the corresponding internal representation of the Elman ASRNN at that particular time step. The network's internal representation at each point is obtained with Sammon Transformation

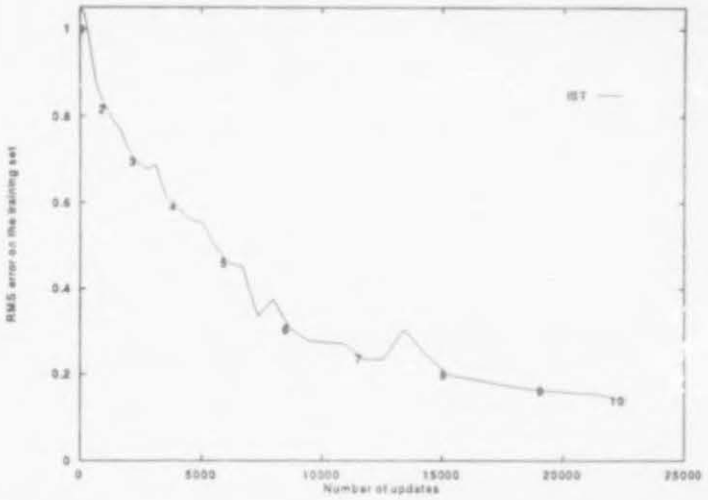


Figure 4.23: Inspection points on RMS error curve for Addition (IST)

Analysis (see section 4.2.6.4). We have used the weight matrices at each inspection point to classify 233 single input patterns of 8-10 column addition. For each point we have extracted the 16 hidden unit activations as the network processed the classification data. Figure 4.24 shows the clusters formed by the projected hidden unit activations at inspection points 1-7 and 10.

Inspection Point	Number of Subsets	Number of Updates	RMS Value	% Correctness on Training Set	% Correctness on Test Set
1	4	308	0.999	0.0%	0.0%
2	8	1103	0.816	28.9%	11.9%
3	12	2327	0.694	37.5%	22.7%
4	16	3993	0.591	59.7%	49.8%
5	20	6099	0.459	74.2%	44.2%
6	24	8681	0.304	87.0%	69.5%
7	28	11717	0.235	93.5%	91.4%
8	32	15243	0.199	94.9%	90.1%
9	36	19264	0.161	97.1%	95.7%
10	39	22582	0.140	98.4%	97.8%

Table 4.6: Simulation results after each inspection point during training of the Elman ASRNN for Addition (IST)

Although the RMS value is still very high (0.999) at inspection point one, Figure 4.24(1) shows that the network has already distinguished between the *done* cluster (ND's and RD's) and the rest of the transitions, which forms a tight cluster. Since the input sequence of the done action

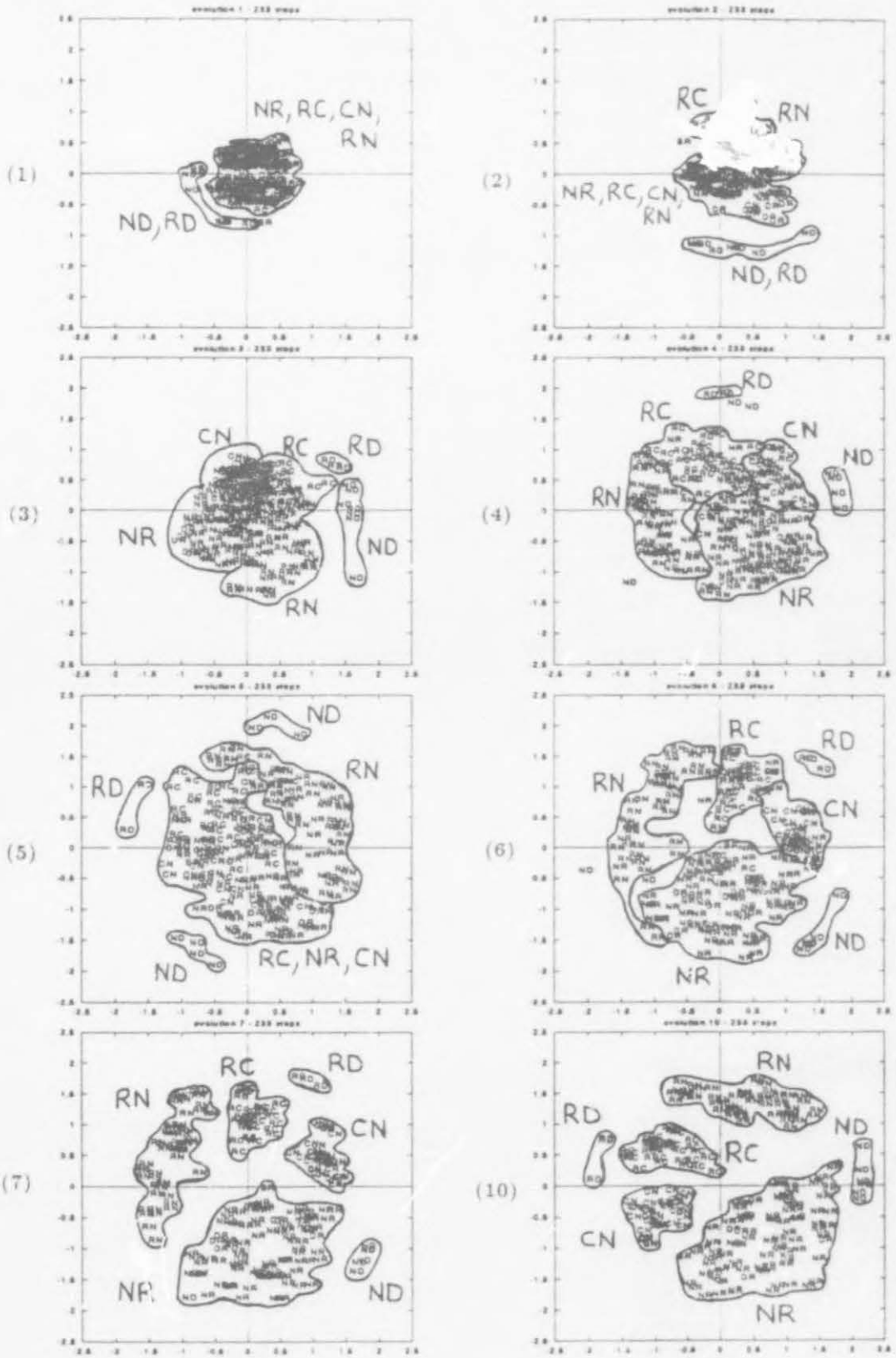


Figure 4.24: Evolution of Sammon Transformation Analysis for Addition (IST)

differs more substantially from the others (for the former the done bit is set to one, whereas it is zero for the others), the network first learns the done transitions. BR, the *Begin-Result* transition, forms a cluster on its own, since it is the starting transition having no previous value (it's location is dependent on the initial network values). At inspection point two, it is not only the *done* transition cluster that is clearly distinguished, but the RN and even a RC cluster start to emerge. After 12 subsets or 2327 updates, at inspection point three, the transitions in the main cluster start to spread outward into four fairly distinct groups representing NR, RN, RC, and CN. At this point the *done* cluster splits into two separate clusters, ND and RD. The network has learned to distinguish between a done action that follows a next action (ND) and one that follows a result action (RD), where the latter is due to a previous carry action. Between inspection point three and four, the network increases its generalization performance on the test set by more than 27%, reaching almost 50% at point four. This result is embodied in Figure 4.24(4), where the main cluster start to evolve into four fairly distinct clusters, although still flawed with misclassifications. We note that the rotation of clusters does not imply anything meaningful about the cluster formation, and is only related to the Sammon Transformation Analysis procedure that applies a random function to the first vector. At inspection point five, the network performance seems to deteriorate as transitions RC, NR, and CN are again merged into a single cluster. This is also reflected in a decrease of 5% in the network's generalization performance between points four and five. However, according to the network's decrease of 0.13 in its RMS value and success percentage increase of 15% on the training set (Table 4.6), the network performance is improving. This seems at first a bit contradictory, but a closer look at Figure 4.24(5) shows that the transitions are actually moving in a more outwardly direction and a larger RN cluster is being formed. The learning curve (Figure 4.23) flattens out between inspection points five and seven, which is reflected in the 47% increase in the generalization performance on the test set (from points five to seven). The large jump in the learning curve between points five and six can be visualized in the Figure 4.24(6), where all six clusters are now discernable. At inspection point seven, which enters the flat part of the learning curve, the six clusters are for the first time well separated without any misclassifications in any of the clusters for the classification data. From inspection points seven to ten the network gradually improves its generalization performance until it reaches 97%, resulting in clearly defined clusters as shown in Figure 4.24(7).

In this section we presented an unique exploration of the learning dynamics of the Elman ASRNN for addition by not only visualizing the evolution of clusters formed by Sammon Transformation Analysis, but also discussing it in terms of a mapping of inspection points on the learning curve, simulation results at each point (number of subsets and updates, RMS value, and the percentage

correctness on the training and test set), and learning curve behaviour. This analysis furthered our understanding of how the network learns to distinguish between different input sequences and how clusters are formed.

4.2.7.3 Visualization of Hidden Activation Time Traces

In this section we visualize the *time traces* or temporal paths of hidden unit activations for particular input of temporal $\overline{XOR}$ (see section 4.2.6.1) during the learning process. We expect to observe the learning dynamics of the Elman ASRNN as the hidden activation values move in the hidden activation space from their initial positions in the hidden space (determined by the initial random weights) to their final positions in the clusters of temporal $\overline{XOR}$ (which have already been identified in section 4.2.6.1). Figure 4.25 illustrates for two hidden units, a visualization of four different time traces, which correspond to four input patterns, labeled 00, 07, 19, and 75 (they were randomly chosen from 100 patterns labeled 00 through to 99). The dots on each time trace denote time-steps. Thus tightly spaced dots indicate slower movement in the hidden activation space, whereas sparsely spaced dots indicate faster movement or more substantial changes in the hidden activations. B, C, and D are the labels of some of the clusters identified in Figure 4.5. The hidden activations for time trace 00 initially started to move quickly in the direction of cluster B, but then made a slow and steady turn away from cluster B and eventually accelerated upwards to its target cluster C. This interesting behaviour can be attributed to both clusters B and C having transitions with output zero. The hidden activations for time trace 75, also destined for cluster C, behaved similarly to time trace 00, except for the fast moving turn in the direction of cluster D, before it finally turned to the right and headed for its target cluster. The hidden activations for time trace 19 moved slowly and deliberately straight into cluster B, before turning sharply upwards in the direction of its designated cluster D. This movement is due to the Elman ASRNN's inability during the early part of the learning to remember the previous input when the current input is one. Only the hidden activations for time trace 07 went straight to its target cluster. All four time traces, emphasize the "strong attraction" of cluster B, which represents transitions with a previous input of zero, current input of one, and output of zero. This can be attributed to the initial random weights that favour cluster B as well as the more general transition that the cluster represents, having an output of zero. The hidden activation values changed very slowly when the time traces were heading in the wrong direction, and very quickly when on the right track (see time traces 00, 19, and 75). This slow movement can be attributed to weights being trapped in local minima. The visualization of the hidden activation time traces provided us with an opportunity to observe

the network's learning difficulties and how it eventually overcomes these difficulties.

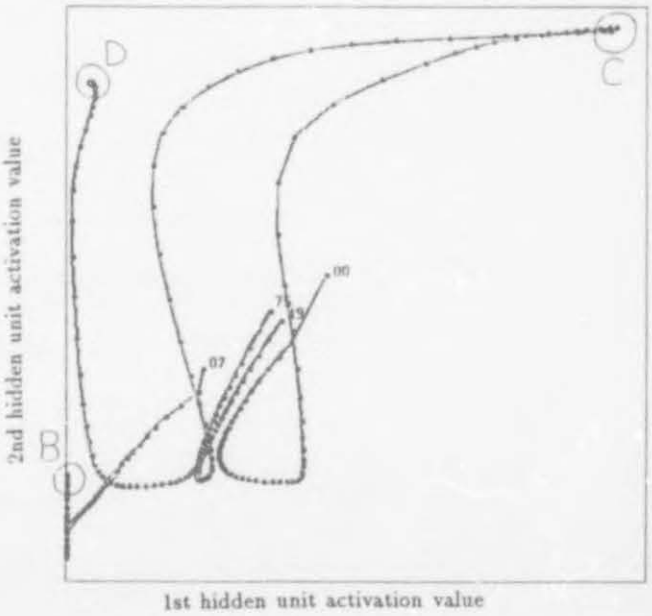


Figure 4.25: Visualization of time traces as hidden activations evolve during the learning process for particular Temporal $\overline{XOR}$ input

4.2.7.4 Learning with Varying Temporal Window Sizes

To counter the effect of previous inputs that are not really relevant for the current output, we have trained the Elman ASRNN with varying temporal window sizes for the detection of two consecutive ones in a bit stream. For this application the network only needs the previous and current input value to determine the current output. The temporal window size is the number of single input patterns from one context reset to another. Context reset (see section 4.2.7.1) means the reset of context unit values and has the effect of removing hidden unit activation history. We started with an initial temporal window size of four, implying a context reset after every four single input patterns. After training the network on window size four, we increased the size to 6, 8, 10, 20 and 50. We finally compared the simulation results with the Elman ASRNN having no context reset (see section 4.2.6.2), implying a temporal window size of 100, since there are 100 single patterns in the training set. The same training and test set as in section 4.1.2 were used in all simulations. The termination criterion was in all cases a 100% correct classification on the test set or the highest possible percentage when the weight changes stabilize. The simulation results for the varying temporal window sizes are summarized in Table 4.7. The Elman ASRNN with temporal window size four found it difficult to learn the task, and after the weight changes stabilized at 2000 updates it only obtained 70% correct classification on the test set. For the

rest of the temporal window sizes (6 to 100), the performance of the network gradually improves from 78% to 100% correct classification on the test set. The Elman ASRNN did not only achieve 100% correct classification for window sizes 50 and 100, but also performed the task in the least number of updates, respectively, 620 and 237. This is also the case for their final RMS values (respectively 0.199 and 0.081). Contrary to our initial expectation that the shorter the memory of the Elman ASRNN (which implies removing irrelevant prior inputs from the input history), the easier is the task, the simulation results of Table 4.7 suggests that the converse holds: the larger the temporal window the easier the Elman ASRNN learns the task. It is almost as if the Elman ASRNN uses the longer hidden activation history more effectively and finds it more informative (about the task at hand) than shorter hidden activation histories. Context reset removes information which were acquired, making it more difficult to learn.

Temporal Window	Final RMS Values	Percentage on Test Set	Number of Updates
4	0.202	70%	2000
6	0.265	78%	1800
8	0.227	80%	1600
10	0.215	89%	1400
20	0.200	96%	1200
50	0.199	100%	620
100	0.081	100%	237

Table 4.7: A performance comparison of the Elman ASRNN with varying temporal windows for the detection of two consecutive ones

We have visualized the hidden activation space for each one of the varying temporal window sizes, where the spaces for sizes 4 to 50 are illustrated in Figure C.5 (Appendix C). For visualization purposes we have used the 100 single input training patterns. The network performance of only 70% on the test set for window size four is reflected in the couple of misclassifications in clusters B and D. For window size six, cluster A is divided between clusters B and C, resulting in a 78% performance on the test set. The clusters formed in the hidden activation space for window size eight also suffers from a few misclassifications. For window size 10, the clusters are well defined, for size 20 even better separated, testifying to the 96% correct classification on the test set, and for sizes 50 and 100 the network performance of 100% is reflected in both cluster formations. Finally, we have also obtained graphical results with Hierarchical Cluster Analysis, which confirm the visualization results for each temporal window size. By examining the Elman ASRNN with different temporal window sizes, we have determined that the longer the hidden

activation history, the better the network's performance.

4.3 Jordan ASRTNs

A Jordan architecture-specific recurrent threshold network with n input units, a single hidden layer of h threshold units, s output threshold units, and a single state layer of s linear units, which contain a linearly averaged copy of previous output values (a proportion of its previous values plus the previous output values), is denoted by *n:h:s Jordan ASRTN*. When $s = 1$, we denote the network by *n:h:1 Jordan ASRTN*. An *n:h:s* Jordan ASRTN can be viewed as an *(n+s):h:s* FFTN having s special additional inputs, the state units.

In this section we look into the number and type of cells formed by a *n:h:s* Jordan ASRTN and compare the results with the corresponding *n:h:s* FFTN and Elman ASRTN.

4.3.1 Number of Cells

We again assume that the h hyperplanes in the first hidden layer are in general position. Let $C(h, m)$ be the number of cells formed by h planes in m-space, where $m = n + s$ is the expanded input dimension $m = n + s$ of an *n:h:s* Jordan ASRTN. The number of cells formed by a Jordan ASRTN is

$$C(h, n+s) = C(h-1, n+s) + C(h-1, n+s-1) \tag{4.9}$$

$$= \sum_{i=0}^{min(h,m)} \frac{h!}{i!(h-i)!}$$

$$= \begin{cases} 2^h & h \leq m = n+s \\ \sum_{i=0}^{n+s} \frac{h!}{i!(h-i)!} & h > m = n+s \end{cases} \tag{4.10}$$

Thus the number of cells formed by h hyperplanes in m-space is independent of the expanded input dimension m of a Jordan ASRTN only when there are less hidden units than input and output units put together. When this condition holds, the number of cells formed is equal to 2^h.

We now consider the case when there are more hidden units than the sum of input and output units, i.e. $h > m = n + s$.

Adding an input or state unit:
From equation (4.10) it is clear that adding an input or state unit while keeping the same number

of hyperplanes increases the number of cells by $\begin{pmatrix} h \\ n+s \end{pmatrix} = \frac{h!}{(n+s)!(h-(n+s))!}$, since

$$C(h, n+s) = C(h, n+s-1) + \begin{pmatrix} h \\ n+s \end{pmatrix} \tag{4.11}$$

The above-mentioned is also relevant when adding an output unit, since this implies adding a state unit (there is an one-to-one correspondence between the output and state units of a Jordan ASRTN).

Adding a hidden unit:

When we add a hidden unit while keeping the same number of input units, the following result is achieved by rewriting equation (4.11) with h replaced by $h-1$, finding the expression for $C(h-1, n+s-1)$, and substituting this in equation (4.9),

$$\begin{aligned} C(h, n+s) &= 2C(h-1, n+s) - \begin{pmatrix} h-1 \\ n+s \end{pmatrix} \\ &= 2\sum_{i=0}^{n+s} \frac{(h-1)!}{i!(h-1-i)!} - \frac{(h-1)!}{(n+s)!(h-1-(n+s))!} \end{aligned} \tag{4.12}$$

As long as $h < n+s$, $C(h, n+s)$ doubles every time we add a hyperplane, i.e. from 2^h to 2^{h+1} (see equation (4.10)). However, for $h > n+s$, equation (4.12) shows that by adding another hyperplane, the number of cells increases to double the amount minus $\begin{pmatrix} h \\ n+s \end{pmatrix}$, which is equal to the number of intersection points generated by $h-1$ hyperplanes intersecting in $n+s$ dimensions.

Adding a hidden and input unit simultaneously:

By substituting for $C(h-1, n+s)$ from (4.11) into equation (4.9), the number of cells is now

$$C(h, n+s) = 2C(h-1, n+s-1) + \begin{pmatrix} h-1 \\ n+s \end{pmatrix} \tag{4.13}$$

4.3.2 Open and Closed Cells

The number of open cells formed by an *n:h:s* Jordan ASRTN is

$$C_o(h, n+s) = \begin{cases} 2^h & h \leq m = n+s \\ 2\sum_{i=0}^{n+s-1} \frac{(h-1)!}{i!(h-1-i)!} & h > m = n+s \end{cases} \tag{4.14}$$

The number of closed cells formed by an *n:h:s* Jordan ASRTN is

$$C_c(h, n+s) = \begin{cases} 0 & h \leq m = n+s \\ \frac{(h-1)!}{(n+s)!(h-1-n-s)!} & h > m = n+s \end{cases} \tag{4.15}$$

For a Jordan ASRTN the expanded input dimension m is only larger than the number of hyperplanes h when $h < n+s$. When this condition holds, there are no closed cells and all the cells are open. As the number of hidden units increases beyond the number of inputs and outputs, the number of closed cells increases.

4.3.3 Imaginary Cells and Disconnected Regions

From equation (4.5) we can derive the number of imaginary cells for Jordan ASRTNS when $h > n+s$, which is

$$C_i(h, n+s) = 2^h - C(h, n+s) = \sum_{i=n+s+1}^{h} \frac{h!}{i!(h-i)!} \tag{4.16}$$

When $h \leq n+s$, $C(h, n+s) = 2^h$, which implies zero imaginary cells and therefore no disconnected decision regions (since cells in a disconnected region can only be connected by imaginary cells). However, when $h > n+s$, Jordan ASRTNs are capable of forming disconnected decision regions (in contrast to Elman ASRTNs), which are connected together through a set of imaginary cells (see Figure 4.27).

4.3.4 Examples

In this section we compare *n:h:s* FFTNs with their corresponding *n:h:s* Jordan and Elman ASRTNs in terms of the number and types of cells formed. We proceed with examples illustrating the two cases, $h \leq n+s$ and $h > n+s$, where $h > n$.

Example 1: Let $n = 1$, $h = 2$, and $s = 1$. Figure 4.26 illustrates a *1:2:1* FFTN and the corresponding *1:2:1* Jordan and Elman ASRTNs in terms of network architecture, input and hidden space. In this example the weights and thresholds of the FFTN are $w_1 = 0.7$, $w_2 = 0.6$, $t_{11} = 0.2$, $t_{12} = 0.5$, $v_1 = -0.7$, $v_2 = 0.8$, and $t_2 = -0.3$. The corresponding values for the Jordan ASRTN are $w_{11} = -0.7$, $w_{12} = 0.6$, $t_{11} = -0.2$, $w_{21} = 0.4$, $w_{22} = 0.9$, $t_{12} = 0.6$, $v_1 = 0.8$, $v_2 = 0.9$, and $t_2 = 0.6$. The corresponding values for the Elman ASRTN are $w_{11} = 0.5$, $w_{12} = 0.8$, $w_{13} = 0.4$, $t_{11} = 0.6$, $w_{21} = 0.5$, $w_{22} = 0.6$, $w_{23} = -0.8$, $t_{12} = -0.3$, $v_1 = 0.8$, $v_2 = 0.9$, and $t_2 = 0.6$. For the FFTN the number of cells is $C(2,1) = \sum_{i=0}^{1} \frac{2!}{i!(2-i)!} = 3$, since

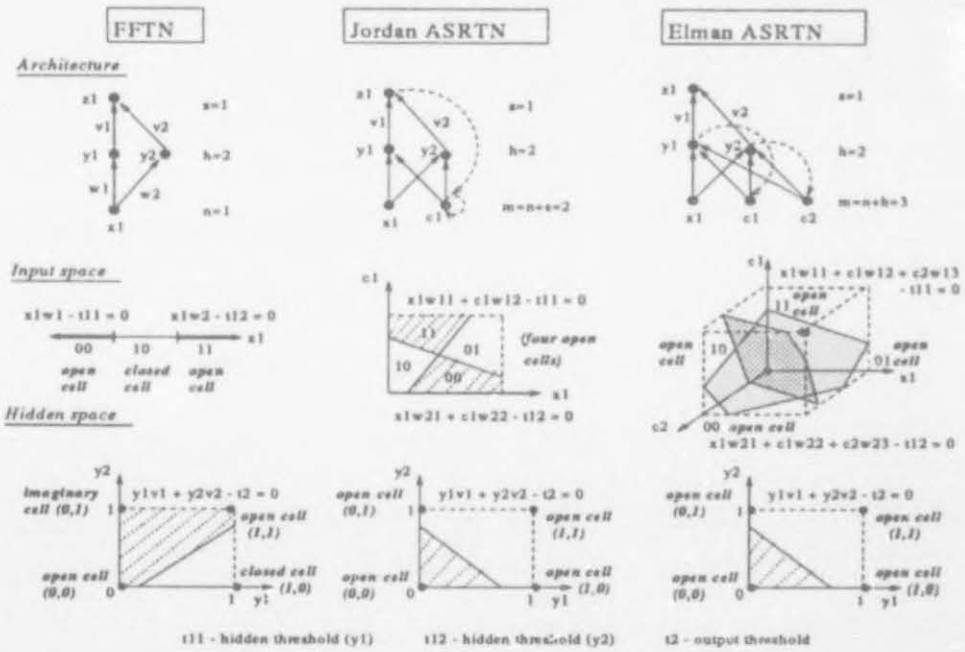


Figure 4.26: Example 1: Architecture, input and hidden space for *1:2:1* FFTN and Jordan and Elman ASRTNs.

$h > n$. For the Jordan ASRTN the number of cells is $C(h, n+s) = 2^h$, since $h < m = n + s$. Thus $C(2,2) = 2^2 = 4$. For the Elman ASRTN the number of cells is $C(h, n+h) = 2^h$, since $h < m = n + h$. Thus $C(2,3) = 2^2 = 4$. The number of open cells for the FFTN is $C_o(2,1) = 2\sum_{i=0}^{0} \frac{(2-1)!}{i!(2-1-i)!} = 2$. The number of open cells for both ASRTNs is $2^h = 2^2 = 4$ and the number of closed cells zero. The number of closed cells formed by the hyperplanes of the *1:2:1* FFTN is $C_c(2,1) = \frac{(2-1)!}{1!(2-1-1)!} = 1$. The number of imaginary cells for the FFTN is $C_i(2,1) = 2^2 - C(2,1) = 4-3 = 1$, while for both ASRTNs it is $C_i(2,3) = 2^2 - C(2,3) = 4-4 = 0$. Two of the three cells in the one-dimensional input space of the FFTN are open (labeled 00 and 11 in Figure 4.26), while the other cell's input variables are bounded by the two hyperplanes (the closed cell is labeled 10). However, in the hidden space of the FFTN four different cells exist. The cell labeled 01 is one which is labeled by a hidden activation value that has no corresponding input values, thus being an imaginary cell. The function of this particular cell is to connect the two open cells in this case to form a disconnected decision region (consisting of the union of the cells labeled 00 and 11). Figure 4.26 illustrates the four open cells in the three-dimensional input space of both *1:2:1* ASRTNs and the two-dimensional hidden space containing zero closed and imaginary cells and four open cells (as opposed to only two open cells for FFTN). This example shows that when $h = n + 1$, and one state unit is added to the input layer, the number of cells and open cells increases to 2^h, whereas the number of closed and imaginary cells decreases to

zero.

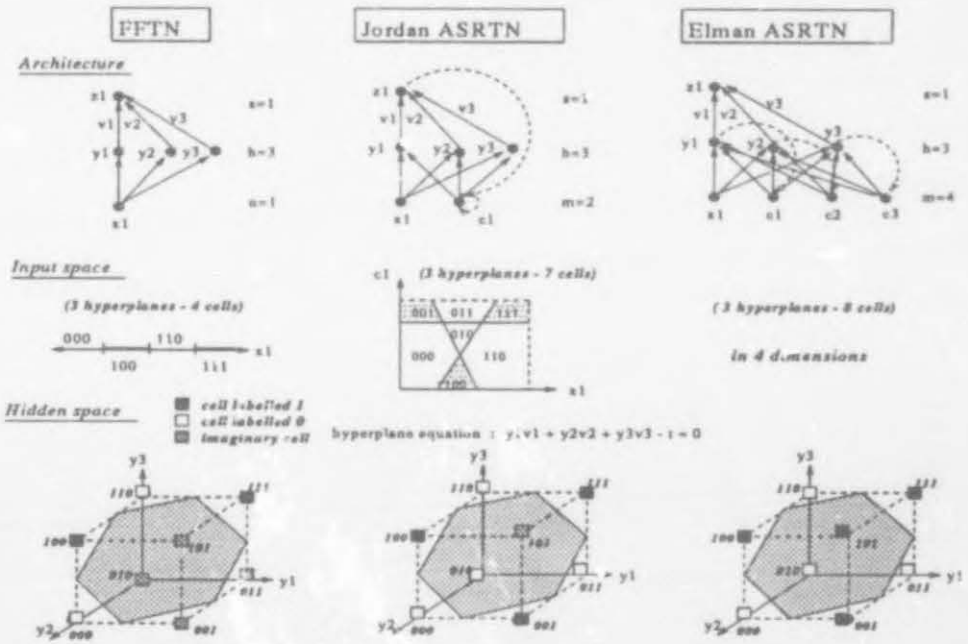


Figure 4.27: Example 2: Architecture, input and hidden space for *1:3:1* FFTN and Jordan and Elman ASRTNs.

Example 2: The smallest Jordan ASRTN that would have not only open cells, but also closed and imaginary cells (and therefore disconnected regions) is an ASRTN with one input, three hiddens, one output, and accordingly one state unit. Thus let $n = 1$, $h = 3$, and $s = 1$. Figure 4.27 illustrates a *1:3:1* FFTN and the corresponding *1:3:1* Jordan and Elman ASRTNs in terms of network architecture, input and hidden space. The hyperplane equation in the hidden space for each network is $y_1v_1 + y_2v_2 + y_3v_3 - t = 0$ with weight and threshold values of $v_1 = 0.3$, $v_2 = 0.35$, $v_3 = 0.4$, $t = 0.5$. Since $h > n$, the number of real cells for the *1:3:1* FFTN is $C(3,1) = \sum_{i=0}^{1} \frac{3!}{i!(3-i)!} = 4$. Since $h > n + s$, the number of real cells for the *1:3:1* Jordan ASRTN is $C(h, n+s) = C(3, 1+1) = \sum_{i=0}^{2} \frac{3!}{i!(3-i)!} = 7$. The number of real cells for the *1:3:1* Elman ASRTN is $C(h, n+h) = C(3, 1+3) = 2^3 = 8$. The number of open cells for the FFTN is $C_o(3,1) = 2\sum_{i=0}^{0} \frac{(3-1)!}{i!(3-1-i)!} = 2$. For the Jordan ASRTN the number of open cells is $C_o(h, n+s) = C_o(3, 1+1) = 2\sum_{i=0}^{1} \frac{(3-1)!}{i!(3-1-i)!} = 6$. For the Elman ASRTN, however, it is $2^h = 2^3 = 8$. The number of closed cells formed by the hyperplanes of the *1:3:1* FFTN is $C_o(3,1) = \frac{(3-1)!}{1!(3-1-1)!} = 2$. For the Jordan ASRTN the number of closed cells is $C_o(3, 1+1) = \frac{(3-2)!}{2!(3-1-2)!} = 1$. For the Elman ASRTN the number of closed cells is zero. The number of imaginary cells for the FFTN is $C_i(3,1) = 2^3 - C(3,1) = 8 - 4 = 4$, while for the Jordan ASRTN it is $C_i(3, 1+1) = 2^3 - C(3,2) = 8 - 7 = 1$, and for the Elman ASRTN it is

$C_i(3, 1+3) = 2^3 - C(3,4) = 8 - 8 = 0$. Two of the four cells in the one-dimensional input space of the FFTN are open (labeled 000 and 111 in Figure 4.27), while the other two cell's input variables are bounded by the three hyperplanes (the closed cells are labeled 100 and 110). However, in the hidden space of the FFTN eight different cells exist. The cells labeled 001, 010, 011, and 101 are cells which are labeled by a hidden activation value that has no corresponding input values, thus being imaginary cells. One of the functions of these cells are to connect real cells in the hidden space forming disconnected decision regions. In this case imaginary cell 101 connects the closed cell 100 and open cell 111 forming a disconnected decision region that corresponds to $z_1 = 1$. On the other hand, the imaginary cell 010 connects the open cell 000 and closed cell 110, both corresponding to an output $z_1 = 0$. There are seven cells in the input space, six of them open and one closed. The state unit of the Jordan ASRTN has the effect of increasing the number of open cells from two to six, thus reducing the closed and imaginary cells to one each. In this case the only imaginary cell 101 connects the three open cells 001, 100, and 111 (all of them corresponding to $z_1 = 1$). Cell 010, which was imaginary in the FFTN's case, is now open and joins the cells corresponding to $z_1 = 0$. The three context units of the Elman ASRTN reduce the number of closed and imaginary cells to zero and increase the number of open cells to eight. Note that cell 101, which in the FFTN and Jordan ASRTN's case was an imaginary cell connecting real cells, is now open and joins the other three open cells 001, 100, and 111 to form a connected decision region that corresponds to $z_1 = 1$. This example shows that when $h > n + 1$ and a state unit is added to the input layer of the *n:h:1* Jordan ASRTN, the number of real cells and open cells increases at the expense of closed and imaginary cells. With a corresponding *n:h:1* Elman ASRTN the number of open cells increases even further to 2^h, whereas the number of closed and imaginary cells decreases to zero.

Example 3: In this example let us consider a case where the Jordan and Elman ASRTNs have an identical number of state and context units, respectively. Let $n = 1$, $h = 3$, and $s = 3$. Figure 4.28 illustrates a *1:3:1* FFTN and the corresponding *1:3:3* Jordan and Elman ASRTNs in terms of network architecture, input and hidden space. For the FFTN the number of cells is $C(3,1) = \sum_{i=0}^{1} \frac{3!}{i!(3-i)!} = 4$, since $h > n$. For the Jordan ASRTN the number of cells is $C(3, 1+3) = 2^3 = 8$, since $h < m = n + s$. For the Elman ASRTN the number of cells is $C(3, 1+3) = 2^3 = 8$, since $h < m = n + h$. The number of open cells for the FFTN is $C_o(3,1) = 2\sum_{i=0}^{0} \frac{(3-1)!}{i!(3-1-i)!} = 2$. The number of open cells for both ASRTNs is $2^h = 2^3 = 8$ and the number of closed cells zero. The number of closed cells formed by the hyperplanes of the FFTN is $C_c(3,1) = \frac{(3-1)!}{1!(3-1-1)!} = 2$. The number of imaginary cells for the FFTN is $C_i(3,1) = 2^3 - C(3,1) = 8-4 = 4$, while for both ASRTNs it is $C_i(3,4) = 2^3 - C(3,4) = 8-8 = 0$. Two of the four cells in the one-dimensional input space of the FFTN are open (labeled 000 and

111), while the other cell's input variables are bounded by the two hyperplanes (the closed cells 100 and 110). In the hidden space of the FFTN eight different cells exist, four of them being imaginary. Figure 4.28 illustrates the eight open cells in the three-dimensional hidden space of both *1:3:3* ASRTNs. This example shows that when $h \leq n + s$ and s state units are added to the input layer, the number of cells and open cells increases to 2^h, whereas the number of closed and imaginary cells decreases to zero.

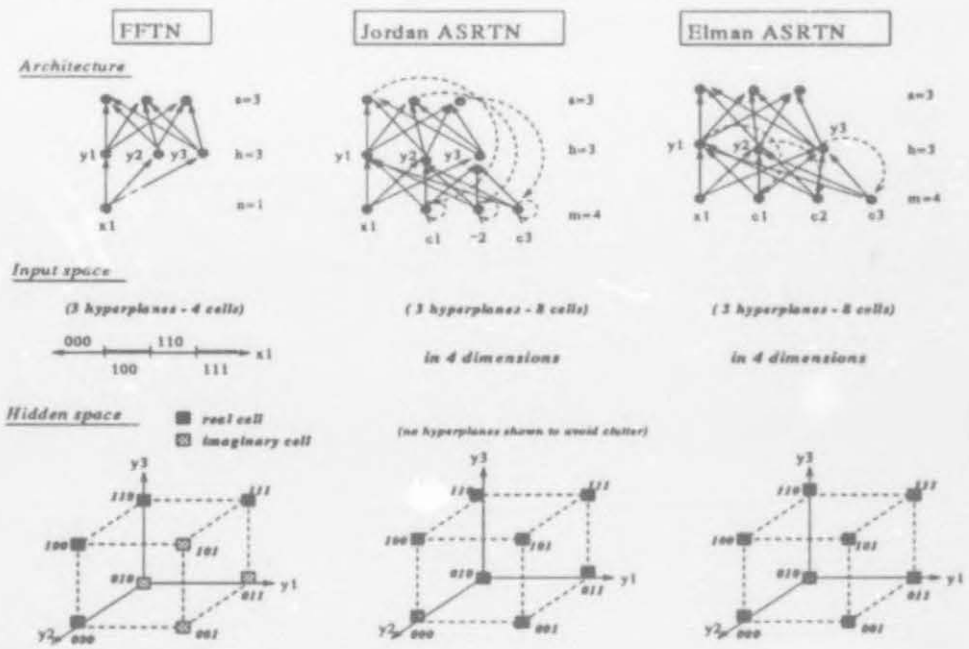


Figure 4.28: Exa- ple 3: Architecture, input and hidden space for *1:3:3* FFTN and Jordan and Elman ASRTNs.

4.3.5 Interpretation of equations

When $h \leq n$, the number of cells for an *n:h:s* FFTN and its corresponding *n:h:s* Jordan ASRTN is $2^h = \sum_{i=0}^{h} \frac{h!}{i!(h-i)!}$. In this case the s additional inputs (state units) of the Jordan ASRTN are not playing any role in the formation of new cells, since the maximum number of cells has already been formed in the n-dimensional input space of the FFTN, namely 2^h. The FFTN does not have any closed or imaginary cells, so the addition of state units to the input units cannot decrease the number of closed and imaginary cells any further.

However, when $h > n$, the state units have a definite effect on the number and type of cells formed, and we can determine the effect accurately when $h \leq n + s$. We next discuss four different cases: $h > n + 1$, $h = n + 1$, $h > n + s$, and $h \leq n + s$. Let us first consider an *n:h:1*

Jordan ASRTN when $h > n+1$. Since the number of real cells formed by the ASRTN in this case is $\sum_{i=0}^{n+1} \frac{h!}{i!(h-i)!}$ and also equal to the number of open cells plus the number of closed cells, the following equations hold

$$\begin{aligned} C(h,n+1) &= \sum_{i=0}^{n+1} \frac{h!}{i!(h-i)!} \\ &= \sum_{i=0}^{n} \frac{h!}{i!(h-i)!} + \frac{h!}{(n+1)!(h-n-1)!} \\ &= 2\sum_{i=0}^{n-1} \frac{(h-1)!}{i!(h-1-i)!} + \frac{(h-1)!}{n!(h-1-n)!} + \frac{(h-1)!}{n!(h-1-n)!} \\ &\quad + \frac{(h-1)!}{(n+1)!(h-n-2)!} \qquad (4.17) \\ &= 2\sum_{i=0}^{n-1} \frac{(h-1)!}{i!(h-1-i)!} + 2\frac{(h-1)!}{n!(h-1-n)!} + \frac{(h-1)!}{(n+1)!(h-n-2)!} \qquad (4.18) \end{aligned}$$

The last two terms of equation (4.17) are derived from

$$\begin{pmatrix} h \\ n+1 \end{pmatrix} = \begin{pmatrix} h-1 \\ n \end{pmatrix} + \begin{pmatrix} h-1 \\ n+1 \end{pmatrix}$$

The number of closed and/or imaginary cells in the FFTN that have become open and closed cells in the Jordan ASRTN is $\frac{h!}{(n+1)!(h-n-1)!}$. Since the last term of equation (4.18) is the number of closed cells in an *n:h:1* Jordan ASRTN, the sum of the first two terms of that equation is the number of open cells in the Jordan ASRTN. The number of imaginary cells for such an ASRTN is equal to $2^h - C(h,n+1) = \sum_{i=n+2}^{h} \frac{h!}{i!(h-i)!}$

The number of real cells formed by the hyperplanes of an *n:h:1* Jordan ASRTN when $h = n+1$ is

$$\begin{aligned} C(h,n+1) &= \sum_{i=0}^{h} \frac{h!}{i!(h-i)!} \\ &= \sum_{i=0}^{n} \frac{h!}{i!(h-i)!} + \sum_{n+1}^{h} \frac{h!}{n!(h-i)!} \\ &= \sum_{i=0}^{n} \frac{h!}{i!(h-i)!} + \frac{h!}{(n+1)!(h-n-1)!} \\ &= \sum_{i=0}^{n} \frac{h!}{i!(h-i)!} + 1 \\ &= 2\sum_{i=0}^{n-1} \frac{(h-1)!}{i!(h-1-i)!} + \frac{(h-1)!}{n!(h-1-n)!} + 1 \qquad (4.19) \end{aligned}$$

With this special network the only imaginary cell in the FFTN case is turned into an open cell for the ASRTN. The term $\frac{(h-1)!}{n!(h-1-n)!}$ gave the number of closed cells for the FFTN, which is now also open cells for the ASRTN.

Let us now consider an *n:h:s* Jordan ASRTN when $h > n + s$. The number of real cells for such a network is

$$\begin{aligned} C(h, n+s) &= \sum_{i=0}^{n+s} \frac{h!}{i!(h-i)!} \\ &= \sum_{i=0}^{n} \frac{h!}{i!(h-i)!} + \sum_{i=n+1}^{n+s} \frac{h!}{(n+1)!(h-n-1)!} \\ &= 2\sum_{i=0}^{n-1} \frac{(h-1)!}{i!(h-1-i)!} + \frac{(h-1)!}{n!(h-1-n)!} + \sum_{i=n+1}^{n+s} \frac{h!}{i!(h-i)!} \end{aligned} \tag{4.20}$$

The term $\sum_{i=n+1}^{n+s} \frac{h!}{i!(h-i)!}$ is the number of closed and imaginary cells in the FFTN case, which are turned into real cells in the Jordan ASRTN.

The number of cells formed by an *n:h:s* Jordan ASRTN when $h \leq n + s$ is

$$\begin{aligned} C(h, n+s) &= \sum_{i=0}^{h} \frac{h!}{i!(h-i)!} \\ &= \sum_{i=0}^{n} \frac{h!}{i!(h-i)!} + \sum_{i=n+1}^{h} \frac{h!}{i!(h-i)!} \\ &= 2\sum_{i=0}^{n-1} \frac{(h-1)!}{i!(h-1-i)!} + \frac{(h-1)!}{n!(h-1-n)!} + \sum_{i=n+1}^{h} \frac{h!}{i!(h-i)!} \end{aligned} \tag{4.21}$$

This equation gives an exact account of the effect of adding s state units to the input layer of a FFTN: the first term $2\sum_{i=0}^{n-1} \frac{(h-1)!}{i!(h-1-i)!}$ was the number of open cells in the FFTN case, which is now still open cells in the Jordan ASRTN case; the second term $\frac{(h-1)!}{n!(h-1-n)!}$ was the number of closed cells for the FFTN, which is now open cells for the ASRTN; the third term $\sum_{i=n+1}^{h} \frac{h!}{i!(h-i)!}$ was the number of imaginary cells for the FFTN, now also being open cells for the Jordan ASRTN.

4.3.6 Capability: Jordan versus Elman ASRTNs

As we have already mentioned, the Jordan ASRTN performs well when the network is supposed to generate output sequences for a constant input, and not so good with tasks where the input is not reflected directly in the output. This inability is due to the recurrent connections coming from the output units and not from the hidden units (which is the case with the Elman ASRTN). The Jordan ASRTN therefore has access to only the current input and the output history. It keeps no record of previous inputs or the internal state of the network. This deficiency of the Jordan network was illustrated by Cottrell and Tsung (1991) when they changed the original output sequence of the addition task (section 4.2.6.4) from "*result - carry - next*" to "*result -*

next - carry". The Jordan network was not able to solve the addition task with the latter output sequence, since after writing the *result*, the *next* action fetches the next pair of digits as input, so that it is not possible for the Jordan ASRTN to determine whether the next step should be a *carry* or not. The Elman ASRTN was able to solve this problem, since it can remember input that is not reflected in its output by having hidden unit activation values, which are a transformed version of the input, recycled at each time step. Is this deficiency of the Jordan network related to the number and types of cells (especially closed and imaginary cells) or is it only related to the recurrent connections coming from the outputs, i.e. the output unit values representing the union of cells in the hidden space? We argue that only the latter is relevant, since by just adding state units until the number of hiddens are smaller or equal to the number of inputs plus the state units, the number of closed and imaginary cells can be reduced to zero (and the open cells increased to 2^h). The state units will still record the output history and the network will still have difficulty remembering inputs not reflected in its output. The outputs will still have values representing unions of cells, independent of how many or type of cells. Therefore the Jordan network's deficiency cannot be related to the number of cells or the types of cells, especially closed and imaginary cells.

However, for a specific problem with a fixed number of input and output variables (i.e. a fixed n and s) and $h > n + s$, an *n:h:s* Elman ASRTN always has more cells in the input space than an *n:h:s* Jordan network (see Figure 4.25 for an example). Even if the number of hidden units is incremented with one for both networks, the number of cells for the Elman network doubles, whereas the number of cells for the Jordan network increases to double the amount minus $\binom{h}{n+s}$ (see equation (4.12)).

4.4 Temporal Autoassociation ASRTNs

A Temporal Autoassociation architecture-specific recurrent threshold network with n input units, a single hidden layer of h threshold units, a single context layer of h linear units, each containing a copy of a corresponding hidden threshold unit value on the previous time step, and $n+h+s$ output threshold units, consisting of n output units computing the estimated input values, h output units computing the estimated context values, and s output units computing the estimated target of the network, is denoted by *n:h:s Temporal Autoassociation ASRTN*. This ASRTN can be viewed as a *(n+h):h:(n+h+s)* FFTN having h special additional inputs and $n+h$ special additional outputs (which approximate the n inputs and h previous hidden values).

Although the Elman ASRTN and its corresponding Temporal Autoassocation ASRTN have a different number of outputs (due to the addition of output units representing the estimated input and context values), they share the same input-context and hidden layer structure, and have a similar number of input, context and hidden units, i.e. they have similar input spaces. Thus the capability results obtained for the Elman ASRTN, i.e. the number of real, open, closed, and imaginary cells, and disconnected regions, also hold for the Temporal Autoassociation ASRTN. However, since there are $n + h$ more outputs in the output layer compared to an *n:h:s* Elman ASRTN, there are $n + h$ extra hyperplanes in the hidden space to assist in grouping the cells in the input space into regions. This would result in finer regions being formed, yielding a finer classification.

4.5 Summary

In this chapter we have investigated the classification capabilities and dynamics of the three main architecture-specific recurrent networks, namely Elman, Jordan, and Temporal Autoassociation networks. The capability analysis was conducted for the threshold version of these networks (threshold activation functions for the hidden and output units). It was indicated that an architecture-specific recurrent threshold network can be viewed as a feedforward threshold network with special additional inputs, the context or state units, whose values are provided by the network itself. The necessary terminology was defined, and for each network the possible types and number of cells in the input and hidden unit activation space were determined. Table 4.8 summarises some of the classification capability results of *n:h:s* Elman, Jordan, and Temporal Autoassociation ASRTNs. We have also given account of the effect of adding context and state units by comparing *n:h:s* FFTNs with their corresponding *n:h:s* ASRTNs in terms of various examples and interpreting the equations for the number of cells. It was also indicated that the conclusions of this analysis can be extended to networks with the well-known sigmoid activation function when the gain parameter is large.

We have also explored the dynamics of the classification process by analyzing the clusters formed by the hidden unit activations of the network. It was shown that an Elman ASRNN can learn to simulate a finite state machine, which was derived from these clusters. The internal representation of the network were obtained by visualizing the input activation space and using clustering techniques, such as Hierarchical Cluster Analysis, Principal Component Analysis, and Sammon Transformation Analysis. The correspondence between the simulated FSM and the one initially constructed for the Addition training data was also determined. For Addition the

Classification Capabilities	Elman and Temporal Autoassociation ASRTNs	Jordan ASRTNs
Number of cells	$C(h, n+h) = 2^h$	$C(h, n+s) = 2^h$ if $h \leq n+s$ $C(h, n+s) = \sum_{i=0}^{n+s} \frac{h!}{i!(h-i)!}$ if $h > n+s$
Open cells	$C_o(h, n+h) = 2^h$	$C_o(\quad n+s) = 2^h$ if $h \leq n+s$ $C_o(h, n+s) = 2\sum_{i=0}^{n+s-1} \frac{(h-1)!}{i!(h-1-i)!}$ if $h > n+s$
Closed cells	$C_c(h, n+h) = 0$	$C_c(h, n+s) = 0$ if $h \leq n+s$ $C_c(h, n+s) = \frac{(h-1)!}{(n+s)!(h-1-n-s)!}$ if $h > n+s$
Imaginary cells	$C_i(h, n+h) = 0$	$C_i(h, n+s) = 0$ if $h \leq n+s$ $C_i(h, n+s) = 2^h - \sum_{i=n+s+1}^{h} \frac{h!}{i!(h-i)!}$ if $h > n+s$
Disconnected regions	none	yes, through a set of imaginary cells

Table 4.8: Classification capabilities of *n:h:s* Elman, Jordan and Temporal Autoassociation ASRTNs

construction of the Mealy machine also enabled us to identify non-deterministic elements in the training data. We have demonstrated with five training strategies that training is much easier (all of them between 34% and 44%) with deterministic data as opposed to non-deterministic data, even though the difference in the two training sets was only 0.6% (percentage of bits flipped). The dynamics of the learning process in ASRNNs was then explored by visualizing the evolution of clusters formed during the learning process. It furthered our understanding of how the network learns to distinguish between different input sequences and how clusters are formed. The visualization of the hidden activation time traces provided us with an opportunity to observe the network's learning difficulties and how it eventually overcomes these difficulties. It was also shown that the network's performance improves with longer hidden unit activation histories. We have further compared the capability of Jordan and Elman networks. It was concluded that the former's inability to deal with tasks where the input is not reflected directly in the output, cannot be related to the number or the types of cells. For the case when $h > n+s$, an *n:h:s* Elman ASRTN has always more cells in the input space than an *n:h:s* Jordan ASRTN. We have finally indicated that the capability results obtained for the Elman network also hold for the Temporal Autoassociation network.

In the next chapter we derive bounds for the capacity and number of hidden units of ASRTNs, where one would find that different results are obtained for Elman, Jordan, and Temporal Autoassociation ASRTNs.

Chapter 5

Bounds for Capacity and Number of Hidden Units

In this chapter the complexity analysis of architecture-specific recurrent neural networks is continued by proving upper and lower bounds for their capacity and number of hidden units. The focus is again on the three main architecture-specific recurrent *threshold* networks, namely Elman, Jordan, and Temporal Autoassociation ASRTNs. By determining bounds for the capacity and the number of hidden units of these networks, questions are adressed such as how many examples a network can remember and how many hidden units are needed to compute an arbitrary mapping. The capacity of an ASRTN is computed as a function of the network architecture, whereas the number of hidden units is obtained as a function of the number of examples as well as the number of input and output units. Bounds for the number of hidden units is important for the optimal design of an ASRTN architecture, since the number of hidden units is in practice chosen experimentally. In particular, for Elman and Temporal Autoassociation ASRTNs the number of context units is dependent on the number of hidden units. We do not only prove upper and lower bounds for *n:h:1* and *n:h:s* ASRTNs, but also compare related bounds and give some examples. Since an ASRTN can be viewed as a feedforward threshold network with special additional inputs, we determine the ASRTN bounds from a feedforward perspective. Apart from proving ASRTN bounds, we have discovered some mistakes in FFTN proofs and made the appropriate corrections. Again the conclusions of this analysis can be extended to networks with the sigmoid activation function in the "high-gain limit".

5.1 Capacity

In this section the focus is on the computational capacity of an ASRTN as a property of the network architecture only, that is independent of the learning algorithm. The capacity of a network is a measure of how much a network can learn. Different measures of capacity could be used, such as the number of examples in general position (definition 4.6) that can be stored by the network, the number of examples that the network can store before the probability of error on recall reaches 1/2, and the Vapnik-Chervonenkis dimension of the class of functions (definition 5.5) realizable by the network. The main concern in this section is with the first two types of capacity, which are respectively called *deterministic* and *probabilistic* capacity. For the probabilistic capacity the examples are chosen randomly and an output of zero or one is randomly assigned to each example. However, for the deterministic capacity the weights and thresholds of the network are obtained deterministically from examples in general position. The deterministic capacity N for a particular type of ASRTN is defined as follows: N is the number of examples in general position if for any assignment of outputs there exists a set of weights and thresholds that realizes the input/output relation and for a set of $N+1$ examples in general position such a set of weights and thresholds cannot be found. We next define a *dichotomy*, which is used for determining the probabilistic capacity of *n:h:1* ASRTNs.

Definition 5.1:
Let F be a class of $\{0,1\}$-valued functions on $\mathcal{R}^n$ and let S be a set of N points in $\mathcal{R}^n$ ($|S| = N$). A *dichotomy* of S induced by $f \in F$ is a set of assignments of 0 or 1 to each point in S [Baum & Hausler, 1989].

In the following definition for *n:h:s* ASRTNs (which is an extension of the previous definition), an *s-bit categorization* of a set is defined, where the number of categories formed by the s output units (having binary values) is 2^s.

Definition 5.2:
A set of assignments of one of $\{0, 1, \ldots, 2^s - 1\}$ to each point in a set S ($|S| = N$), is called an *s-bit categorization* of S, where the categories are denoted by integers between 0 and $2^s - 1$ inclusive [Sakurai, 1993].

We proceed by proving upper and lower bounds for the deterministic and probabilistic capacity of *n:h:1* and *n:h:s* Elman, Jordan, and Temporal Autoassociation ASRTNs. The bounds obtained for these networks along with the corresponding bounds for feedforward networks are summarized in the final section of this chapter.

5.1.1 Elman ASRTNs

Definition 5.3:

An *n:h:1 Elman ASRTN* is said to *realize* a function $F : \mathcal{R}^{n+h} \rightarrow \{0, 1\}$ if, for any n-dimensional external input and h-dimensional context input corresponding to a point x of $\mathcal{R}^{n+h}$, the *n:h:1* Elman ASRTN outputs a value equal to $F(x)$.

Definition 5.4:

An *n:h:s Elman ASRTN* is said to *realize* an s-bit categorization $F : \mathcal{R}^{n+h} \rightarrow \{0, \ldots, 2^s - 1\}$ if, for any n-dimensional external input and h-dimensional context input corresponding to a point x of $\mathcal{R}^{n+h}$, the *n:h:s* Elman ASRTN outputs a value equal to $F(x)$.

Consider an Elman ASRTN with n inputs, a single hidden layer of h threshold units, a single context layer of h units (each containing a copy of the corresponding previous hidden state value), and s output units. The context units operate on the same level as the input units and receive their values via one-to-one fixed connections from the hidden layer. On a specific time step they act as additional inputs and together with the input units activate the hidden units. Thus the *n:h:s* Elman ASRTN operates similarly to an *(n+h):h:s* feedforward threshold network, having an augmented input dimension of $m = n + h$.

In order to determine bounds for the deterministic capacity of feedforward threshold networks, Sakurai (1992) assumed that the N input vectors are in general position. According to definition 4.6, a set of N vectors is in general position in n-dimensional space if every subset of n or fewer vectors is linearly independent. To determine the deterministic capacity of Elman ASRTNs the N input-context vectors must also be in general position in *(n+h)*-dimensional space, since an *n:h:s* Elman ASRTN operates similarly to an *(n+h):h:s* feedforward threshold network where the input-context vectors are the augmented *(n+h)*-dimensional input vectors. We prove this precondition by assuming that the N input vectors are in general position and then proving that the input-context vectors are in general position in *(n+h)*-dimensional space. The following lemma is used to prove that an *n:h:s* Elman ASRTN with a set of N n-dimensional input vectors in general position has any n or fewer *(n+h)*-dimensional input-context vectors linearly independent. This result will enable us to prove that the N input-context vectors are also in general position in *(n+h)*-dimensional space.

Lemma 5.1:

If an *n:h:s* Elman ASRTN has a set of N n-dimensional input vectors in general position, then any n or fewer *(n+h)*-dimensional input-context vectors are linearly independent.

Proof:

Since the n-dimensional input vectors of an *n:h:s* Elman ASRTN are in general position, every subset of n or fewer vectors is linearly independent (see definition 4.6). Suppose we have the following set of N input vectors:

$$\begin{aligned} \{ \quad \vec{v_1} &= (a_{11}, a_{12}, \ldots, a_{1n}), \\ \vec{v_2} &= (a_{21}, a_{22}, \ldots, a_{2n}), \\ &\vdots \\ \vec{v_N} &= (a_{N1}, a_{N2}, \ldots, a_{Nn}) \ \} \end{aligned} \tag{5.1}$$

where a_{ij} $(i = 1 \ldots N$ and $j = 1 \ldots n)$ is a scalar and $\vec{v_i}$ is a vector, then an *n:h:s* Elman ASRTN has the following set of *(n+h)*-dimensional input-context vectors:

$$\begin{aligned} \{ \quad \vec{w_1} &= (a_{11}, \ldots, a_{1n}, c_{11}, \ldots, c_{1h}), \\ \vec{w_2} &= (a_{21}, \ldots, a_{2n}, c_{21}, \ldots, c_{2h}), \\ &\vdots \\ \vec{w_N} &= (a_{N1}, \ldots, a_{Nn}, c_{N1}, \ldots, c_{Nh}) \ \} \end{aligned} \tag{5.2}$$

where $c_{il} = \begin{cases} 0 \\ 1 \end{cases}$ $(i = 1 \ldots N$ and $l = 1 \ldots h)$ and $\vec{w_i}$ is a vector. We have to prove that *every* subset of n or fewer vectors from the set $\{\vec{w_1}, \ldots, \vec{w_N}\}$ is linearly independent. Since any subset of a linearly independent set is linearly independent [Lipschutz, 1968], we only need to prove that any subset of n vectors from the set $\{\vec{w_1}, \ldots, \vec{w_N}\}$ is linearly independent.

By using the principle of mathematical induction, we begin by proving for $h = 1$ that every subset of n vectors from the set

$$\begin{aligned} A = \{ \quad \vec{w_1} &= (a_{11}, \ldots, a_{1n}, c_{11}), \\ &\vdots \\ \vec{w_N} &= (a_{N1}, \ldots, a_{Nn}, c_{N1}) \ \} \end{aligned}$$

is linearly independent. For any selection of n vectors $\vec{y_1}, \ldots, \vec{y_n}$ where $\vec{y_i} \in A$ $(i = 1 \ldots n)$, it must be proved that $p_1 = p_2 = \ldots = p_n = 0$ for the following equation

$$p_1\vec{y_1} + p_2\vec{y_2} + \ldots + p_n\vec{y_n} = \vec{0} \tag{5.3}$$

where p_j $(j = 1 \ldots n)$ is a scalar. Equation (5.3) leads to the following homogeneous system of linear equations

$$p_1 a_{11} + p_2 a_{21} + \ldots + p_n a_{n1} = 0$$
$$\vdots$$
$$p_1 a_{1n} + p_2 a_{2n} + \ldots + p_n a_{nn} = 0 \tag{5.4}$$

which is n equations in n unknowns, and the following equation

$$p_1 c_{11} + p_2 c_{21} + \ldots + p_n c_{n1} = 0 \tag{5.5}$$

Equations (5.4) and (5.5) are then a homogeneous system of $n+1$ equations in n unknowns, which has only the trivial solution ($p_1 = p_2 = \ldots = p_n = 0$) or infinitely many nontrivial solutions solutions in addition to the trivial solution. The homogeneous system of linear equations can be solved by Gauss-Jordan elimination and has the following augmented matrix

$$\begin{bmatrix} a_{11} & a_{21} & \ldots & a_{n1} & 0 \\ & & \vdots & & \\ a_{1n} & a_{2n} & \ldots & a_{nn} & 0 \\ c_{11} & c_{21} & \ldots & c_{n1} & 0 \end{bmatrix}$$

Since every subset of n vectors from the set (5.1) is linearly independent, the first n rows of the augmented matrix can be reduced to row-echelon form to produce the following matrix

$$\begin{bmatrix} 1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 1 & 0 & \ldots & 0 & 0 \\ & & \vdots & & & \\ 0 & 0 & 0 & \ldots & 1 & 0 \\ c_{11} & c_{21} & c_{31} & \ldots & c_{n1} & 0 \end{bmatrix}$$

Since every nonzero value of $c_{il} = \begin{cases} 0 \\ 1 \end{cases}$ $(i = 1 \ldots N \text{ and } l = 1)$ can be reduced to zero by subtracting the $(n+1)$'th row from one of the first n rows, we obtain

$$\begin{bmatrix} 1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 1 & 0 & \ldots & 0 & 0 \\ & & \vdots & & & \\ 0 & 0 & 0 & \ldots & 1 & 0 \\ 0 & 0 & 0 & \ldots & 0 & 0 \end{bmatrix}$$

for any of the 2^n possible combinations of binary values for $c_{11}, \ldots, c_{n1}$. This matrix only produces the trivial solution $p_1 = p_2 = \ldots = p_n = 0$.

Assume now for $h = k$ that every subset of n vectors from the set

$$\begin{cases} \vec{w_1} = (a_{11}, \ldots, a_{1n}, c_{11}, \ldots, c_{1k}), \\ \vdots \\ \vec{w_N} = (a_{N1}, \ldots, a_{Nn}, c_{N1}, \ldots, c_{Nk}) \end{cases}$$

is linearly independent. That is, for any selection of n vectors the following homogeneous system of linear equations,

$$\begin{aligned} p_1 a_{11} + p_2 a_{21} &+ \ldots + p_n a_{n1} = 0 \\ &\vdots \\ p_1 a_{1n} + p_2 a_{2n} &+ \ldots + p_n a_{nn} = 0 \\ p_1 c_{11} + p_2 c_{21} &+ \ldots + p_n c_{n1} = 0 \\ &\vdots \\ p_1 c_{1k} + p_2 c_{2k} &+ \ldots + p_n c_{nk} = 0 \end{aligned} \tag{5.6}$$

only produce $p_1 = p_2 = \ldots = p_n = 0$ as a solution.

We now need to prove for $h = k + 1$ that the following homogeneous system of linear equations

$$\begin{aligned} p_1 a_{11} + p_2 a_{21} &+ \ldots + p_n a_{n1} = 0 \\ &\vdots \\ p_1 a_{1n} + p_2 a_{2n} &+ \ldots + p_n a_{nn} = 0 \\ p_1 c_{11} + p_2 c_{21} &+ \ldots + p_n c_{n1} = 0 \\ &\vdots \\ p_1 c_{1k} + p_2 c_{2k} &+ \ldots + p_n c_{nk} = 0 \end{aligned} \tag{5.7}$$

which is $n+k$ equations in n unknowns, and the following equation

$$p_1 c_{1(k+1)} + p_2 c_{2(k+1)} + \ldots + p_n c_{n(k+1)} = 0 \tag{5.8}$$

only produce $p_1 = p_2 = \ldots = p_n = 0$ as a solution.

Equations (5.7) and (5.8) are a homogeneous system of $n+k+1$ equations in n unknowns, which has only the trivial solution ($p_1 = p_2 = \ldots = p_n = 0$) or infinitely many nontrivial solutions solutions in addition to the trivial solution. The homogeneous system of linear equations can

be solved by Gauss-Jordan elimination and has the following augmented matrix

$$\begin{bmatrix} a_{11} & a_{21} & \dots & a_{n1} & 0 \\ & & \vdots & & \\ a_{1n} & a_{2n} & \dots & a_{nn} & 0 \\ c_{11} & c_{21} & \dots & c_{n1} & 0 \\ & & \vdots & & \\ c_{1k} & c_{2k} & \dots & c_{nk} & 0 \\ c_{1(k+1)} & c_{2(k+1)} & \dots & c_{n(k+1)} & 0 \end{bmatrix}$$

Since the equations (5.6) of the induction step, which have an augmented matrix consisting of the first $n+k$ rows of the former matrix, only produce $p_1 = p_2 = \ldots = p_n = 0$ as a solution, the first $n+k$ rows can be reduced to row-echelon form to produce the following matrix

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ & & \vdots & & & \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \\ & & \vdots & & & \\ 0 & 0 & 0 & \dots & 0 & 0 \\ c_{1(k+1)} & c_{2(k+1)} & \dots & c_{n(k+1)} & 0 \end{bmatrix}$$

Since every nonzero value of $c_{i(k+1)} = \begin{cases} 0 \\ 1 \end{cases}$ $(i = 1 \ldots n)$ can be reduced to zero by subtracting the $(n+k+1)$'th row from one of the first n rows, we obtain

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ & & \vdots & & & \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \\ & & \vdots & & & \\ 0 & 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

for any of the 2^n possible combinations of binary values for $c_{1(k+1)}, \ldots, c_{n(k+1)}$. This matrix only produce the trivial solution $p_1 = p_2 = \ldots = p_n = 0$. By the principle of mathematical induction, it follows that every subset of n vectors from the set $\{\vec{w}_1, \ldots, \vec{w}_N\}$ is linearly independent for all

$h \geq 1$. Therefore an *n:h:s* Elman ASRTN has any n or fewer *(n+h)*-dimensional input-context vectors linearly independent. $\triangle$

The previous lemma showed that if an *n:h:s* Elman ASRTN has a set of N input vectors in general position, it also has any n or fewer input-context vectors linearly independent. The next lemma uses this result to prove that the N input-context vectors are in general position in *(n+h)*-dimensional space. Suppose we have any subset A_1 of n *(n+h)*-dimensional input-context vectors and a disjoint subset A' of h *(n+h)*-dimensional input-context vectors, where $n+h \leq N$ and $h \leq n$. Thus $A' \cap A_1 = \emptyset$. Since Lemma 5.1 showed that every subset of n or fewer input-context vectors are linearly independent, h has to be smaller than or equal to n for the disjoint subset A' of h input-context vectors to be linear independent. By proving that the N input-context vectors so obtained are also in general position in *(n+h)*-dimensional space, the result can be applied to Sakurai's (1992) deterministic capacity bound to obtain a corresponding bound for Elman ASRTNs.

Lemma 5.2:

Suppose an *n:h:s* Elman ASRTN has a set of N input vectors in general position in n-dimensional space. For any subset of n input-context vectors and a disjoint subset of h input-context vectors where $n + h \leq N$ and $h \leq n$, the N input-context vectors are also in general position in *(n+h)*-dimensional space.

Proof:

If an *n:h:s* Elman ASRTN has a set of N n-dimensional input vectors in general position in n-dimensional space, then according to Lemma 5.1 any n or fewer *(n+h)*-dimensional input-context vectors are linearly independent. Suppose we have the following set of N *(n+h)*-dimensional input-context vectors

$$\begin{aligned} \{ \quad \vec{w_1} &= (a_{11}, \ldots, a_{1n}, c_{11}, \ldots, c_{1h}), \\ \vec{w_2} &= (a_{21}, \ldots, a_{2n}, c_{21}, \ldots, c_{2h}), \\ &\vdots \\ \vec{w_N} &= (a_{N1}, \ldots, a_{Nn}, c_{N1}, \ldots, c_{Nh}) \ \} \end{aligned} \tag{5.9}$$

where $c_{il} = \begin{cases} 0 \\ 1 \end{cases}$ $(i = 1 \ldots N$ and $l = 1 \ldots h)$ and $\vec{w_i}$ is a vector. Thus any subset of n or fewer *(n+h)*-dimensional vectors from the set $\{\vec{w_1}, \ldots, \vec{w_N}\}$ is linearly independent. Since $h \leq n$, every subset of h *(n+h)*-dimensional vectors is also linearly independent. Suppose we have any selection of n *(n+h)*-dimensional vectors $\vec{y_1}, \ldots, \vec{y_n}$ and h *(n+h)*-dimensional vectors $\vec{z_1}, \ldots, \vec{z_h}$ from the set (5.9), where $A_1 = \{\vec{y_1}, \ldots, \vec{y_n}\}$, $A' = \{\vec{z_1}, \ldots, \vec{z_h}\}$, $A_1 \cap A' = \emptyset$ and $(n+h) \leq N$. Let

$A_2 = A_1 \cup A'$. A_2 is then a set of $n+h$ $(n+h)$-dimensional input-context vectors. Since $A_1 \subset A_2$, the union $A_2 = A_1 \cup A'$ of linearly independent sets is also linearly independent [Lipschutz, 1968]. Since any subset of a linearly independent set is linearly independent [Lipschutz, 1968], every subset of $n+h$ or fewer input-context vectors from the set $\{\vec{w_1}, \ldots, \vec{w_N}\}$ is linearly independent. Therefore an *n:h:s* Elman ASRTN has its $(n+h)$-dimensional input-context vectors in general position in $(n+h)$-dimensional space. $\triangle$

5.1.1.1 n:h:1 Lower bound

Sakurai (1992) showed that an *n:h:1* feedforward threshold network can store correctly at least $nh+1$ examples in general position in n-dimensional space. This lower bound for the *deterministic capacity* is proved in a constructive manner so that the weights and thresholds are obtained deterministically from examples. In order to obtain a lower bound for the deterministic capacity of an *n:h:1* Elman ASRTN we apply the previous lemma to Sakurai's FFTN bound.

Theorem 5.1:

An Elman ASRTN with n inputs, a single hidden layer of h threshold units, a single context layer of h previous hidden layer values, and one output threshold unit can store at least $(n+h)h+1$ examples in general position, where $h \leq n$.

Proof:

From Lemma 5.2 it follows that an *n:h:1* Elman ASRTN has its N input-context vectors in general position in $(n+h)$-dimensional space if the N input vectors are in general position in n-dimensional space, where $h \leq n$. Sakurai (1992) proved that $N = nh+1$ is a lower bound for the capacity of an *n:h:1* FFTN if it is assumed that the input vectors are in general position in n-dimensional space. Thus, by using similar arguments as Sakurai, a lower bound for the capacity of an *n:h:1* Elman ASRTN can be proved by regarding it as an *(n+h):h:1* FFTN with the augmented input-context vectors as its $(n+h)$-dimensional input vectors. Therefore, by substituting n with $n+h$ in Sakurai's proof, a capacity lower bound of $(n+h)h+1$ examples in general position is obtained for *n:h:1* Elman ASRTNs. Thus an *n:h:1* Elman ASRTN can store at least $(n+h)h+1$ examples in general position in $(n+h)$-dimensional space. $\triangle$

For the temporal $X\overline{OR}$ example (in Chapter 4), an Elman ASRTN with 2 input units, a single hidden layer of 2 threshold units and 2 context units, and one output threshold unit can store at least $(2+2)2+1 = 9$ examples in general position.

5.1.1.2 n:h:1 Upper bound

In order to determine an upper bound for the *probabilistic capacity* of *n:h:1* Elman ASRTNs, let us first obtain an upper bound for the probabilistic capacity of a single threshold unit. Suppose there are N random input vectors in general position, each randomly assigned an output zero or one. These can be regarded as a set S of examples $S_1, \ldots, S_N$ in n-dimensional space. Cover (1965) obtained an upper bound by counting the number of ways of dividing S into two subsets by a hyperplane and using this to determine the probability of an arbitrary division of S being inducible by a hyperplane. A dichotomy (definition 5.1) is a division of S by a hyperplane, with the convention that two hyperplanes which divide the examples into the same subsets induce the same dichotomy and the dichotomies are oriented (i.e. the order of two subsets defined by the dichotomy is taken into account). Let $D(N, n)$ denote the number of dichotomies of N examples of a set S in n dimensions, where the total number of divisions of S into two subsets is 2^N. The probability of an arbitrary division of S for a threshold unit having a non-zero threshold is $D(N, n+1)/2^N$ (where the non-zero threshold gives an extra dimension). The probabilistic capacity for a single threshold unit is that N for which the probability $D(N, n+1)/2^N = 1/2$ which occurs when $N = 2(n+1)$. Cover (1965) and Mitchison & Durbin (1989) showed that

$$D(N, n+1) = 2\sum_{i=0}^{n} \binom{N-1}{i} \tag{5.10}$$

Therefore the probability $D(N, n+1)/2^N$ can be expressed as the sum of the binomial expansion of $(1/2 + 1/2)^{N-1}$ up to the n-th term, which becomes an $1/2$ when $N = 2(n+1)$. Mitchison & Durbin used this probability condition to prove an upper bound of $O(-h \log_2 h)$ examples for an *n:h:1* FFTN with threshold units having a threshold of zero.

The number of dichotomies of N examples in general position in $n+h$ dimensions for a single threshold hidden unit with a non-zero threshold in an *n:h:1* Elman ASRTN is therefore $D(N, n+h+1)$, where $n+h \leq N$ and $h \leq n$. In order to obtain an upper bound for the probabilistic capacity of *n:h:1* Elman ASRTNs we count the number of distinct ways of separating examples by sets of h hyperplanes in $n+h$ dimensions. Any set of h hyperplanes will induce a partition and there is a unique partition induced by each set of h dichotomies. There are $D(N, n+h+1)^h$ such sets. But not all the partitions in this manner obtained will be distinct, since it is possible for different sets of dichotomies to yield the same partition. Thus the N for which $\frac{D(N,n+h+1)^h}{2^N} = 1/2$ or $D(N, n+h+1)^h = 2^{(N-1)}$ gives an upper bound for these ASRTNs. By using this probability condition in a similar fashion as Mitchison & Durbin, except for counting the number of dichotomies for h hyperplanes in $n+h$ dimensions where the threshold units have non-zero

thresholds, an upper bound for an *n:h:1* Elman ASRTN is obtained next.

Theorem 5.2:

The *probabilistic capacity* of an Elman ASRTN with n inputs, a single hidden layer of h threshold units, a single context layer of h previous hidden layer values and one threshold output unit is at most $O((n+h+1)h\log_2 h)$ examples in general position, where $n+h \leq N$, $h \leq n$ and $N \geq (2+\sqrt{2})(n+h+1)+2$.

Proof:

The objective in this proof is to obtain that N for which $D(N, n+h+1)^h = 2^{(N-1)}$ gives an upper bound capacity for an *n:h:1* Elman ASRTN. Similar to Mitchison & Durbin, we replace $D(N, n+h+1)$ by larger expressions to eventually obtain an upper bound for which $D(N, n+h+1) \geq 2^{N-1}$. This will also be an upper bound for a solution to $D(N, n+h+1)^h = 2^{N-1}$, since for small N, $D(N, n+h+1)^h > 2^{N-1}$. By applying Mitchison & Durbin's steps to $D(N, n+h+1)$, which are outlined in Appendix D.1.1(a), we obtain

$$D(N, n+h+1) < \begin{pmatrix} N \\ n+h+1 \end{pmatrix}. \tag{5.11}$$

The next step is to rewrite $\begin{pmatrix} N \\ n+h+1 \end{pmatrix}$. as an expression without any factorials and to prove the following inequality (see Appendic D.1.1(b))

$$\begin{pmatrix} N-1 \\ n+h \end{pmatrix} < \{e^{1/12} \cdot (N/(2\pi(n+h+1)(N-n-h-1)))^{1/2}\} \cdot \frac{N^{n+h+1}}{\{(n+h+1)^{n+h+1} \cdot (1-(n+h+1)/N)^{N-n-h-1}\}} \tag{5.12}$$

Since $2^{-1/h} > e^{1/12} \cdot (N/(2\pi(n+h+1)(N-n-h-1))^{1/2}$ and applying inequality (5.12), the following result holds,

$$\begin{pmatrix} N \\ n+h+1 \end{pmatrix} < \{2^{-1/h} \cdot N^{n+h+1}/\{(n+h+1)^{n+h+1} \cdot (1-(n+h+1)/N)^{N-n-h-1}\} \tag{5.13}$$

Since $D(N, n+h+1)^h = 2^{N-1}$ and it was proved that $D(N, n+h+1) < \begin{pmatrix} N \\ n+h+1 \end{pmatrix}$, the next inequality holds (following the steps outlined in Appendix D.1.1(c))

$$(N/(n+h+1)h)\log_e 2 \;<\; \log_e(N/(n+h+1)) - (N/(n+h+1)-1)\log_e(1-1/(N/(n+h+1))) \quad (5.14)$$

By following the arguments in Appendix D.1.1(d) the inequality (5.14) can be rewritten as $ej = eh\log_2(ej)$, where $j = N/(n+h+1)$. The equation $ej = eh\log_2(ej)$ has the solution $ej = eh\log_2(eh\log_2(eh\ldots))$, the nesting being infinite [Mitchison *et al*, 1989]. Since $j=N/(n+h+1)$ we can rewrite the solution as $N = (n+h+1)h\log_2(eh\log_2(eh\ldots))$, which leads to $N = O((n+h+1)h\log_2 h)$. The nested expression converges, because the sequence $\{b_n\}$ defined by $b_{n+1} = x\log_2 b_n$ increases and is bounded above by x^2 if $x > 4$ [Mitchison *et al*, 1989]. The condition $N \geq (2+\sqrt{2})(n+h+1)+2$ is satisfied, provided that $h \geq 2$. Thus $N = O((n+h+1)h\log_2 h)$ is an upper bound for a solution to $D(N, n+h+1)^h = 2^{N-1}$. $\triangle$

In the next section we define and briefly discuss an alternative measure for the capacity of *n:h:1* Elman ASRTNs.

5.1.1.3 n:h:1 VC Dimension

The Vapnik-Chervonenkis dimension of the class of functions realizable by an ASRTN is also used as a measure of capacity.

Definition 5.5:
A class F of $\{0,1\}$-valued functions defined on a set S is said to *shatter* a finite set $X \subseteq S$ if, for each of the $2^{|X|}$ classifications of the points in X, there is a function in F that computes the classification. The *Vapnik-Chervonenkis (VC) dimension* of F, denoted by *VC-dim(F)*, is the size (cardinality) of the largest subset X of S that F shatters [Vapnik, 1982].

Only loose bounds of the VC dimension of FFTNs are known. Let F be the class of all functions computed by FFTNs with n inputs, w weights and thresholds, and h hidden threshold units. Then *VC-dim(F)* is lower bounded by $nh+1$ [Sakurai, 1993] and upper bounded by $2w\log_2(eh)$ [Baum & Hausler, 1989]. Similarly, let G be the class of all functions computed by Elman ASRTNs with n inputs, w weights and thresholds, and h hidden and h context units. Then $(n+h)h+1 \leq$ *VC-dim(G)* $\leq 2w\log_2(eh)$, where the lower bound follows from Theorem 5.1.

5.1.1.4 n:h:s Upper bounds

In this section an upper bound for the deterministic capacity of an *n:h:s* FFTN is corrected, before upper bounds for the deterministic and probabilistic capacity of an *n:h:s* Elman ASRTN are determined.

Sakurai & Yamasaki (1992) proved the following upper bound for the deterministic capacity of *n:h:s* FFTNs:

$$n(\lfloor\frac{h}{2}\rfloor - (P-2) + \lceil(\lfloor\frac{h}{2}\rfloor - 1)/(P-1)\rceil + 1) + (P-1) \geq n(\lfloor\frac{h}{2}\rfloor - P) + P, \tag{5.15}$$

where $P(\leq 2^s)$ is the minimum number of different kinds of associated output vectors. Unfortunately, we discovered some erroneous statements and mistakes in the proof and pointed it out in personal communication to Sakurai. The upper bound should be

$$n(\lfloor\frac{h}{2}\rfloor - (P-2) + \lceil(\lfloor\frac{h}{2}\rfloor + 1)/(P-1)\rceil - 1) + (P-1) > n(\lfloor\frac{h}{2}\rfloor - P) + P \tag{5.16}$$

where the right hand side of the inequality holds for $h \geq 2P$. We proceed by briefly sketching Sakurai & Yamasaki's proof and along the way pointing out mistakes and making the appropriate corrections. The first part of the proof consists of finding hyperplanes and defining weights and thresholds for them in a constructive way: Assume h is even and set $h' = h/2$. The set of examples is divided into P subsets (each consisting of examples with the same associated output vector), which in turn is divided into sub-subsets (each consisting of n examples), except for the largest subset, which is treated as a background subset. For each subset a hyperplane is found on which their examples lie, and then two new hyperplanes close and parallel to the old hyperplane and uniquely on its positive side. The weights and thresholds of the hidden threshold units are defined to designate the new hyperplanes. The next step is to find for each output unit all the sub-subsets corresponding to the target output vectors whose element corresponding to the output unit is one. Finally, the weights and threshold of each output threshold unit are defined, calculating the majority function of hidden threshold unit values corresponding to the sub-subsets, and the minority function for the excluded background subset. The number of examples is a minimum when one of the remaining $P-1$ subsets has a multiple of n examples and each of the remaining $P-2$ subsets has one plus a multiple of n examples. Sakurai & Yamasaki stated inappropriately that the remaining $P-2$ subsets each has only one example. The total number of examples in non-background subsets is $n(h'-(P-2))+(P-2)$, since the number of non-background n-example sub-subsets (those containing n examples each) is $(h'-(P-2))$, the number of examples in n-example sub-subsets is therefore simply $n(h'-(P-2))$, and the number of one-example sub-subsets (those containing one example each) is $(P-2)$. The number

of examples is also a minimum when the minimum of the cardinality of the background subset is the maximum of the cardinality of the non-background subsets when the latter is set as small as possible. The minimum is attained when all the "multiples of n" are distributed as evenly as possible among non-background subsets. Let X be the smallest integer greater or equal to the number of n-example sub-subsets per non-background subset, $X = \lceil (h' - (P-2))/(P-1) \rceil$. Sakurai & Yamasaki's statement "the number of examples in the background subset is at least $\lceil (h' - (P-2))/(P-1) \rceil = \lceil (h'-1)/(P-1) \rceil + 1$" is wrong. The number of examples in the background subset should be $n\lceil (h' - (P-2))/(P-1) \rceil + 1 = n(\lceil (h'+1)/(P-1) \rceil - 1) + 1$. Multiplication by n and addition of 1 were dropped and $+1$ and -1 were reversed by mistake. The minimum is therefore attained when for the non-background subsets:

- $P-2$ subsets have $nX+1$ or $n(X-1)+1$ examples
- one subset has nX or $n(X-1)$ examples

Therefore the background subset has

- $nX+1$ examples when $X+X+(P-3)(X-1)+(P-2) \leq h'$; X for a *multiple-of-n-example* subset, another X for a largest *multiple-of-n-plus-1-example* subset, $(P-3)(X-1)$ for other *multiple-of-n-plus-1-example* subsets, and $(P-2)$ for the *1-example* sub-subsets. The sum of these terms is less than or equal to h' to account for the case when more than one of the $P-2$ subsets have each $nX+1$ examples (in the above-mentioned it is assumed that $P-3$ subsets have each $n(X-1)+1$ examples).
- nX examples when $X+(X-1)+(P-3)(X-1)+(P-2) = h'$; the second term $(X-1)$ is for a largest *multiple-of-n-plus-1-example* subset, while the other terms have similar meaning as in the first case.

When the first case applies an upper bound for the capacity is therefore the sum of $n(h'-(P-2))$ and $nX+1$, which is equal to $n(\lfloor \frac{h}{2} \rfloor - (P-2) + \lceil (\lfloor \frac{h}{2} \rfloor + 1)/(P-1) \rceil - 1) + (P-1)$. When the second case applies, which was overlooked by Sakurai & Yamasaki, an upper bound for the capacity is the sum of $n(h' - (P-2))$ and nX, which is equal to

$$n(\lfloor \frac{h}{2} \rfloor - (P-2) + \lceil (\lfloor \frac{h}{2} \rfloor + 1)/(P-1) \rceil - 1) + (P-2)$$

Sakurai & Yamasaki also incorrectly stated that the left hand side of (5.15) is "greater than or equal to" the right hand side. This should only be "greater than", since for $h \geq 2P$ the following

equations and inequalities hold for the first case:

$$n(\lfloor\frac{h}{2}\rfloor - (P-2) + \lceil(\lfloor\frac{h}{2}\rfloor + 1)/(P-1)\rceil - 1) + (P-1)$$

$$\begin{aligned} &\geq n(\lfloor\frac{h}{2}\rfloor - P + 3) + (P-1) \\ &= n\lfloor\frac{h}{2}\rfloor - nP + 3n + P - 1 \\ &= n(\lfloor\frac{h}{2}\rfloor - P) + 3n + P - 1 \\ &> n(\lfloor\frac{h}{2}\rfloor - P) + P \end{aligned} \tag{5.17}$$

If $h \geq 2P$, the value $n(\lfloor\frac{h}{2}\rfloor - P) + P$ is in both cases an upper bound of the deterministic capacity of *n:h:s* FFTNs.

In order to obtain a related result for an upper bound on the *deterministic capacity* of *n:h:s* Elman ASRTNs, we use similar arguments as in the feedforward case, proved by Sakurai & Yamasaki (1992).

Theorem 5.3:

An Elman ASRTN with n inputs, a single hidden layer of h threshold units, a single context layer of h previous hidden layer values, and s output threshold units can correctly store at least

$$(n+h)(\lfloor\frac{h}{2}\rfloor - (P-2) + \lceil(\lfloor\frac{h}{2}\rfloor + 1)/(P-1)\rceil - 1) + (P-1) > (n+h)(\lfloor\frac{h}{2}\rfloor - P) + P$$

examples in general position, where $h \leq n$, $P(\leq 2^s)$ is the minimum number of different kinds of associated output vectors, and the right hand side of the inequality holds for $h \geq 2P$.

Proof:

According to definition 5.4, an *n:h:s* Elman ASRTN realizes an s-bit categorization $F : \mathcal{R}^{n+h} \rightarrow \{0, \ldots, 2^s - 1\}$ if, for any n-dimensional external input and h-dimensional context input corresponding to a point x of $\mathcal{R}^{n+h}$, the *n:h:s* Elman ASRTN outputs a value equal to $F(x)$. From Lemma 5.2 it follows that an *n:h:s* Elman ASRTN has its N input-context vectors in general position in *(n+h)*-dimensional space if the N input vectors are in general position in n-dimensional space, where $h \leq n$. Sakurai & Yamasaki (1992) proved a lower bound (of which the corrected version is (5.16)) for the capacity of an *n:h:1* FFTN if it is assumed that the input vectors are in general position in n-dimensional space. Thus, by using similar arguments as Sakurai & Yamasaki (with the appropriate corrections as stated above), a lower bound for the capacity of an *n:h:s* Elman ASRTN can be proved by regarding it as an *(n+h):h:s* FFTN with the augmented input-context vectors as its *(n+h)*-dimensional input vectors. Therefore, by replacing n with

$n+h$ in the Sakurai & Yamasaki's proof, a capacity lower bound of

$$(n+h)(\lfloor\frac{h}{2}\rfloor-(P-2)+\lceil(\lfloor\frac{h}{2}\rfloor+1)/(P-1)\rceil-1)+(P-1)>(n+h)(\lfloor\frac{h}{2}\rfloor-P)+P$$

examples in general position is obtained for *n:h:s* Elman ASRTNs, where $h \leq n$ and $P(\leq 2^s)$ is the minimum number of different kinds of associated output vectors. $\triangle$

Suppose we have for the addition application an *8:8:3* Elman ASRTN with $P = 3$. Since $h \geq 2P = 6$, the upper bound of the deterministic capacity of this network is $(8+8)1+4 = 20$ examples in general position.

Mitchison & Durbin (1989) proved that $O(n(1+h/s)\log_2(1+h/s)$ is an upper bound for the probabilistic capacity of *n:h:s* FFTNs. We now prove, in a similar manner, an upper bound which is an order estimation of the probabilistic capacity of *n:h:s* Elman ASRTNs.

Theorem 5.4:

The *probabilistic capacity* of an Elman ASRTN with n inputs, a single hidden layer of h threshold units, a single context layer of h previous hidden layer values, and s output threshold units is at most $N = O((n+h+1)(1+h/s)\log_2(1+h/s)$ examples in general position, where $h \leq n$.

Proof:

From section 5.1.1.2, it follows that the outputs of the h threshold hidden units associated with each of the N $(n+h)$-dimensional input vectors form a vector with h components, and there are at most $D(N, h+1)$ dichotomies of these N h-dimensional vectors. Each of these dichotomies defines a possible output function for the *n:h:s* Elman ASRTN. Since there are s output threshold units, the number of combinations of output functions are at most $D(N,h+1)^s$. There are therefore at most $D(N,h+1)^s$ ways assigning output functions for each partition defined by the hidden units. Thus the total number of distinct outputs which the *n:h:s* Elman ASRTN can generate is at most $D(N,n+h+1)^h \cdot D(N,h+1)^s$. By applying Mitchison & Durbin's arguments to *n:h:s* Elman ASRTNs (see Appendix D.1.2) we obtain the following upper bound

$$N \leq (n+h+1)(1+h/s)\log_2(e(1+h/s)\ldots)) = O((n+h+1)(1+h/s)\log_2(1+h/s)) \quad (5.18)$$

for the probabilistic capacity of these networks, where $h \leq n$. $\triangle$

5.1.2 Jordan ASRTNs

Definition 5.6:

An *n:h:1 Jordan ASRTN* is said to *realize* a function $F : \mathcal{R}^{n+1} \rightarrow \{0,1\}$ if, for any n-dimensional

external input and one-dimensional state input corresponding to a point x of $\mathcal{R}^{n+1}$, the *n:h:1* Jordan ASRTN outputs a value equal to $F(x)$.

Definition 5.7:

An *n:h:s Jordan ASRTN* is said to *realize* an s-bit categorization $F : \mathcal{R}^{n+h} \rightarrow \{0, \ldots, 2^s - 1\}$ if, for any n-dimensional external input and s-dimensional state input corresponding to a point x of $\mathcal{R}^{n+h}$, the *n:h:s* Jordan ASRTN outputs a value equal to $F(x)$.

An *n:h:s* Jordan ASRTN has an s-dimensional state layer, which operates on the same level as the n-dimensional input vector. On a specific time step they act as additional inputs and together with the input units activate the hidden units. Thus the *n:h:s* Jordan ASRTN operates similarly to an *(n+s):h:s* feedforward threshold gate network, having an extended input dimension of $m = n + s$. For the theoretical analysis of Jordan ASRTNs we assume that the state vector at time t is the output vector value at time $t - 1$. The following lemma shows that an *n:h:s* Jordan ASRTN with a set of N n-dimensional input vectors in general position also has any n or fewer *(n+s)*-dimensional input-state vectors linearly independent. This result will enable us to show that the N input-state vectors are also in general position in *(n+s)*-dimensional space.

Lemma 5.3:

If an *n:h:s* Jordan ASRTN has a set of N n-dimensional input vectors in general position, then any n or fewer *(n+s)*-dimensional input-state vectors are linearly independent.

Remark:

By using the principle of mathematical induction in a similar fashion as in the proof of Lemma 5.1 for Elman ASRTNs and substituting h with s, it can be proved that every subset of n vectors from the set $\{\vec{w_1}, \ldots, \vec{w_N}\}$ is linearly independent for all $s \geq 1$. Therefore an *n:h:s* Jordan ASRTN has any n or fewer *(n+s)*-dimensional input-state vectors linearly independent. $\triangle$

Lemma 5.3 showed that if an *n:h:s* Jordan ASRTN has a set of N input vectors in general position, it has any n or fewer input-state vectors linearly independent. To determine the deterministic capacity of Jordan ASRTNs, the next lemma shows that the N input-state vectors are also in general position in *(n+s)*-dimensional space.

Lemma 5.4:

Suppose an *n:h:s* Jordan ASRTN has a set of N input vectors in general position in n-dimensional space. For any subset of n input-state vectors and a disjoint subset of s input-state vectors where $n+s \leq N$ and $s \leq n$, the N input-state vectors are also in general position in *(n+s)*-dimensional

space.

Remark:

This lemma follows directly from Lemma 5.2 for Elman ASRTNs by substituting h with s. $\triangle$

5.1.2.1 n:h:1 Lower bound

In this section a lower bound for the *deterministic capacity* of an *n:h:1* Jordan ASRTN is obtained by applying Lemma 5.4 to Sakurai (1992)'s FFTN bound.

Theorem 5.5:

A Jordan ASRTN with n inputs, a single hidden layer of h threshold units, a single state unit, and one output threshold unit can store at least $(n+1)h+1$ examples in general position.

Remark:

From Lemma 5.4 it follows that an *n:h:1* Jordan ASRTN has its input-state vectors in general position in *(n+1)*-dimensional space if the input vectors are in general position in n-dimensional space, where $n \geq 1$ and the number of input vectors in general position is greater than or equal to $n+1$. By using similar arguments as Sakurai (1992), a lower bound for the capacity of an *n:h:1* Jordan ASRTN can be proved by regarding it as an *(n+1):h:1* FFTN with the augmented input-state vectors as its *(n+1)*-dimensional input vectors. Therefore, by substituting n for $n+1$ in Sakurai's proof, a capacity lower bound of $(n+1)h+1$ examples in general position is obtained for *n:h:1* Jordan ASRTNs. Thus an *n:h:1* Jordan ASRTN can store at least $(n+1)h+1$ examples in general position in *(n+1)*-dimensional space. $\triangle$

5.1.2.2 n:h:1 Upper bound

This section shows an upper bound for the probabilistic capacity of an one-hidden layer Jordan ASRTN with one output threshold unit.

Theorem 5.6:

The *probabilistic capacity* of a Jordan ASRTN with n inputs, a single hidden layer of h threshold units, a single state unit, and one threshold output is at most $O((n+2)h\log h)$ examples in general position, where $N \geq (2+\sqrt{2})(n+2)+2$.

Remark:

The number of dichotomies of N examples in $n+1$ dimensions for a single threshold hidden unit

in an *n:h:1* Jordan ASRTN is $D(N, n+1+1) = D(N, n+2)$. By following similar steps as in the proof of Theorem 5.2 and replacing $n+h+1$ with $n+2$ in $D(N, n+h+1)$, we obtain an upper bound for the capacity of $N = O((n+2)h\log_2 h)$. The condition $N \geq (2+\sqrt{2})(n+2)+2$ is satisfied, provided that $h \geq 2$. Thus $N = O((n+2)h\log_2 h)$ is an upper bound for a solution to $D(N, n+2)^h = 2^{N-1}$. $\triangle$

5.1.2.3 n:h:1 VC Dimension

Let G be the class of all functions computed by Jordan ASRTNs with n inputs, w weights and thresholds, and h hidden and one state unit. Then $(n+1)h+1 \leq VC\text{-}dim(G) \leq 2w\log_2(eh)$, where the lower bound follows from Theorem 5.5. Sakurai (1993) proved a better upper bound for *n:h:1* feedforward networks, namely $nh(\log_2 h + 2\log_2(\log_2 h))$. Likewise we proceed by proving a better upper bound for *n:h:1* Jordan ASRTNs.

Theorem 5.7:

The VC dimension of an n:h:1 Jordan ASRTN is at most $(n+1)h(\log_2 h + 2\log_2(\log_2 h))$ when $n+1 \geq 32$ and $h \geq 256$.

Proof:

From definition 5.5, it follows that if it is proven that for any set of N points the number of dichotomies of the set realizable by an *n:h:1* Jordan ASRTN is less than 2^N, N is an upper bound of the VC dimension of such ASRTNs. In the proof of Theorem 5.6 we have deduced that the number of dichotomies of N points in $n+1$ dimensions for h threshold hidden units in an *n:h:1* Jordan ASRTN is $D(N, n+1+1)^h = D(N, n+2)^h$. There are at most $D(N, h+1)$ dichotomies of the N h-dimensional hidden unit vectors, each one of them defining a possible output function for the *n:h:1* Jordan ASRTN. Since there are only one output threshold unit, the number of combinations of output functions are at most $D(N, h+1)^1$. Thus the total number of distinct outputs which the ASRTN can generate is at most $D(N, n+2)^h \cdot D(N, h+1)^1$. Following the steps outlined in Appendix D.1.3(a) the next inequality holds [Sakurai, 1993]

$$D(N, n+2) \quad < \quad 3 \cdot (e\frac{N}{n+1})^{n+1} \tag{5.19}$$

If we set $N = (n+1)h(\log_2 h)(1 + (2\log_2\log_2 h)/(log_2 h))$, then following the steps in Appendix D.1.3(b) the next inequality holds [Sakurai, 1993]

$$\log_2 N \quad < \quad \log_2(n+1)h + \log_2(\log_2 h) + 1 \tag{5.20}$$

since $\log_2(1 + (2\log_2\log_2 h)/(\log_2 h)) < log_2 2 = 1$. The next step is to prove that $D(N, n+2)^h \cdot D(N, h+1)^1 < 2^N$ or that $\log_2 D(N, n+2)^h \cdot D(N, h+1)^1 - N < 0$. In Appendix

D.1.3(c) we prove this by using the inequalities (5.19) and (5.20) in a similar manner as Sakurai (1993). Thus $N = (n+1)h(\log_2 h + 2\log_2(\log_2 h))$ is an upper bound of the VC dimension of *n:h:1* Jordan ASRTNs when $n+1 \geq 32$ and $h \geq 256$. $\triangle$

5.1.2.4 n:h:s Upper bounds

We first show an upper bound for the deterministic capacity and then an order estimation of the upper bound for the probabilistic capacity of *n:h:s* Jordan ASRTNs.

Theorem 5.8:

A Jordan ASRTN with n inputs, a single hidden layer of h threshold units, a single state layer of s units, and s output threshold units can correctly store at least

$$(n+s)(\lfloor\frac{h}{2}\rfloor - (P-2) + \lceil(\lfloor\frac{h}{2}\rfloor + 1)/(P-1)\rceil - 1) + (P-1) > (n+s)(\lfloor\frac{h}{2}\rfloor - P) + P$$

examples in general position, where $s \leq n$, $P \leq 2^s$, and the right hand side holds for $h \geq 2P$.

Remark:

According to definition 5.7, an *n:h:s* Jordan ASRTN realizes an s-bit categorization $F : \mathcal{R}^{n+s} \rightarrow \{0, \ldots, 2^s - 1\}$ if, for any n-dimensional external input and s-dimensional state input corresponding to a point x of $\mathcal{R}^{n+s}$, the *n:h:s* Jordan ASRTN outputs a value equal to $F(x)$. From Lemma 5.4 it follows that an *n:h:s* Jordan ASRTN has its input-state vectors in general position in *(n+s)*-dimensional space if the input vectors are in general position in n-dimensional space, where $s \leq n$ and the number of input vectors in general position is greater than or equal to $n+s$. The remainder of the proof is similar to that of Theorem 5.3 when substituting $n+s$ for $n+h$ in all the steps and arguments. When the first case applies, an upper bound for the capacity of *n:h:s* Jordan ASRTN is $(n+s)(\lfloor\frac{h}{2}\rfloor - (P-2) + \lceil(\lfloor\frac{h}{2}\rfloor + 1)/(P-1)\rceil - 1) + (P-1)$. When the second case applies (i.e. one less example), an upper bound for the capacity is $(n+s)(\lfloor\frac{h}{2}\rfloor - (P-2) + \lceil(\lfloor\frac{h}{2}\rfloor + 1)/(P-1)\rceil - 1) + (P-2)$. In both cases $(n+s)(\lfloor\frac{h}{2}\rfloor - P) + P$ is an upper bound of the capacity of *n:h:s* Jordan ASRTNs when $h \geq 2P$. $\triangle$

Suppose for the addition application we have a *6:16:6* Jordan ASRTN with P=6. Since $h \geq 2P = 12$, the upper bound for the deterministic capacity is $(6+6)(8-6)+6 = 30$ examples in general position.

The next theorem gives an upper bound for the probabilistic capacity of an *n:h:s* Jordan ASRTN.

Theorem 5.9:

The *probabilistic capacity* of a Jordan ASRTN with n inputs, a single hidden layer of h threshold units, a single state layer of s units, and s output threshold units is at most $O((n+s+1)(1+h/s)\log_2(1+h/s))$ examples in general position, where $s \leq n$.

Remark:

Following the same steps as in the proof of Theorem 5.4 and substituting $n+s+1$ for $n+h+1$, gives the following upper bound

$$N \leq (n+s+1)(1+h/s)\log_2(e(1+h/s)\log_2(e(1+h/s)\ldots)) = O((n+s+1)(1+h/s)\log_2(1+h/s)) \tag{5.21}$$

for the probabilistic capacity of *n:h:s* Jordan ASRTNs, where $s \leq n$. $\triangle$

5.1.3 n:h:s Temporal Autoassociation ASRTNs : Upper bounds

Definition 5.8:

An *n:h:s Temporal Autoassociation ASRTN* is said to *realize* an *(n+h+s)*-bit categorization $F : \mathcal{R}^{n+h} \rightarrow \{0, \ldots, 2^{n+h+s}-1\}$ if, for any n-dimensional external input and h-dimensional context input corresponding to a point x of $\mathcal{R}^{n+h}$, the *n:h:s* Temporal Autoassociation ASRTN outputs a value equal to $F(x)$.

The following lemma shows that an *n:h:s* Temporal Autoassociation ASRTN has its N *(n+h)*-dimensional input-context vectors in general position in n-dimensional space.

Lemma 5.5

If an *n:h:s* Temporal Autoassociation ASRTN with N n-dimensional input vectors in general position, then any n or fewer *(n+h)*-dimensional input-context vectors are linearly independent.

Remark:

Since *n:h:s* Elman and Temporal Autoassociation ASRTNs share the same input-context and hidden layer structure, and have a similar number of input, context and hidden units, the proof of this lemma follows directly from Lemma 5.1. $\triangle$

To determine an upper bound for the deterministic capacity of a Temporal Autoassociation ASRTN, the following lemma shows that the N input-context vectors are also in general position in *(n+h)*-dimensional space.

Lemma 5.6:

Suppose an *n:h:s* Temporal Autoassociation ASRTN has a set of N input vectors in general position in n-dimensional space. For any subset of n input-context vectors and a disjoint subset of h input-context vectors where $n + h \leq N$ and $h \leq n$, the N input-context vectors are also in general position in $(n+h)$-dimensional space.

Remark:

The proof follows directly from Lemma 5.2. $\triangle$

Theorem 5.10:

A Temporal Autoassociation ASRTN with n inputs, a single hidden layer of h threshold units, a single context layer of h previous hidden layer values, and $n + h + s$ output threshold units can correctly store at least

$$(n+h)(\lfloor\frac{h}{2}\rfloor - (P-2) + \lceil(\lfloor\frac{h}{2}\rfloor + 1)/(P-1)\rceil - 1) + (P-1) > (n+h)(\lfloor\frac{h}{2}\rfloor - P) + P$$

examples in general position, where $h \leq n$, $P \leq 2^{n+h+s}$, and the right hand side holds for $h \geq 2P$.

Remark:

The proof of this theorem follows directly from Theorem 5.3 (for Elman ASRTNs), except that $P \leq 2^{n+h+s}$. $\triangle$

Theorem 5.11:

The probabilistic capacity of a Temporal Autoassociation ASRTN with n inputs, a single hidden layer of h threshold units, a single context layer of h previous hidden layer values, and $n+h+s$ output threshold units is at most $O((n+h+1)(1+h/(n+h+s))\log_2(1+h/(n+h+s))$ examples in general position, where $h \leq n$.

Remark:

The proof follows directly from Theorem 5.4. $\triangle$

5.2 Number of hidden units

The purpose of this section is to derive bounds for the number of hidden units of Elman, Jordan, and Temporal Autoassociation ASRTNs. These bounds provide much needed answers to the question of how many hidden units are enough to realize an arbitrary function for a particular

ASRTN. The bounds obtained for these ASRTNs along with the corresponding bounds for feedforward threshold networks are summarized in the final section of this chapter. The FFTN literature on bounds for the number of hidden units include [Baum, 1988], [Baum & Hausler, 1989], [Cosnard *et al*, 1992], [Sakurai, 1992], and [Sakurai & Yamasaki, 1992]. Sakurai showed that $\lceil (N-1)/n \rceil$ hidden units are necessary to realize a specified input/output relation with an *n:h:1* and *n:h:s* FFTN. Sakurai & Yamasaki pointed out that the bound, N/n, obtained by Baum, is wrong, and that it should be $2\lceil N/(2n) \rceil$ (which is a worse bound than $\lceil (N-1)/n \rceil$).

Instead of only giving an expression for the number of hidden units in terms of the number of examples N, we now use an alternative parameter, the minimum distance δ between the two classes. Cosnard *et al* defined the distance between two distinct classes, S^+ and S^-, as $\delta = min\{distance(x,y) | x \in S^+, y \in S^-\}$, where the *distance* is the Euclidean distance in $[0,1]^n$.

Cosnard *et al* (1992) showed that an arbitrary dichotomy (S^+, S^-) can be separated by $n(\lfloor \frac{1}{\delta} \rfloor + 1)$ hyperplanes of a feedforward threshold network. However, this lower bound should in fact be $n(\lfloor \frac{\sqrt{n}}{\delta} \rfloor + 1)$. We proceed by briefly sketching Cosnard *et al*'s proof for a one hidden layer feedforward threshold network and along the way pointing out mistakes and making the appropriate corrections. We consider an arbitrary dichotomy (S^+, S^-) in an n-dimensional unit hypercube. Cosnard *et al* constructed the hyperplanes of the network's hidden layer in the following manner, $x_i = b_j, 1 \leq i \leq n, 1 \leq j \leq \lfloor 1/\delta \rfloor$, with $b_j = j\delta$ and n the number of input units. The hyperplanes divide the unit hypercube in $(\lfloor 1/\delta \rfloor + 1)^n$ cells. It is assumed for the moment that no point of the set S lies on any of the hyperplanes. Cosnard *et al* then incorrectly deduced that if any two points x and y in S belong to the same cell, the $distance(x,y) < \delta$ and the points are therefore in the same class. However, the $distance(x,y)$ can only be smaller than δ if the hyperplanes are constructed in the following manner, $x_i = b_j, 1 \leq i \leq n, 1 \leq j \leq \lfloor 1/d \rfloor$, with $b_j = jd$, where $d = \delta/\sqrt{n}$. Since the distance between two consecutive hyperplanes in each dimension must remain smaller or equal to d, the maximum Euclidean distance between any two points in a cell is $\sqrt{d^2 + d^2 + \ldots + d^2}$ (where d^2 is added n times), which is equal to $\sqrt{nd^2} = d\sqrt{n}$. For example, when $n = 2$, the maximum distance is $\sqrt{d^2 + d^2} = \sqrt{2d^2} = d\sqrt{2}$. Therefore, if any two points x and y belong to the same cell, the $distance(x,y) < d\sqrt{n} = \delta$ and the points are therefore in the same class, i.e. the hyperplanes separate S^+ and S^-. However, in general some points of the set might lie on some hyperplanes. To account for this case, we translate the hyperplanes slightly toward the origin so that all points belong to the interior of cells by subtracting a value ϵ, with $0 \leq \epsilon \leq d$, from all the hyperplane equations. One additional hyperplane for each dimension might be needed, since the distance between two consecutive hyperplanes in each dimension must remain smaller or equal to d. This would result in a total

of $\lfloor 1/d \rfloor + 1$ hyperplanes at most per dimension. Since there are n dimensions, the number of hyperplanes is $n(\lfloor 1/d \rfloor + 1) = n(\lfloor \sqrt{n}/\delta \rfloor + 1)$.

Cosnard *et al* also proved that an arbitrary dichotomy of $[0,1]^2$ cannot be realized by a *2:h:1* FFTN with less than $2/\delta$ hidden units [Cosnard *et al*, 1992]. The more precise lower bound should in fact be $h \geq 2\lceil \sqrt{2}/\delta \rceil$. We proceed by briefly sketching Cosnard *et al*'s proof for a one hidden layer FFTN with one output unit and along the way show the errors and the corrections. The hidden layer of the *n:h:1* FFTN performs a linear separation. If we consider the unit square $[0,1]^2$ with $1 + \lceil 1/d \rceil$ points of a set S on each edge, then each point is alternatively in S^- and S^+. Since there are $1 + \lceil 1/d \rceil$ points on each edge, there are $\lceil 1/d \rceil$ segments on each edge (the number of parts between points on an edge). Since there are 4 edges, the total number of segments linking the points of S to their nearest neigbours of the opposite class is $4\lceil 1/d \rceil$. Thus Cosnard *et al* incorrectly stated that the number of segments is $4/\delta$. Since a straight line cannot cut more than two links and all links must be cut to have a dichotomy, the number of straight lines needed is at least $\frac{4\lceil 1/d \rceil}{2} = 2\lceil 1/d \rceil$ and not $2/\delta$. Therefore $2\lceil 1/d \rceil = 2\lceil \sqrt{2}/\delta \rceil$ is a lower bound for the number of hidden units in a two-dimensional input space to realize an arbitrary dichotomy.

Cosnard *et al* also proved that an arbitrary dichotomy of $[0,1]^n$ cannot be realized by an *n:h:1* FFTN with less than $n/(e\delta)$ hidden units [Cosnard *et al*, 1992]. The lower bound should be $h \geq \lceil (n/e)\lfloor \sqrt{n}/\delta \rfloor \rceil$ as indicated in Appendix D.2.1.

5.2.1 Elman ASRTNs

In this section we derive lower and upper bounds for the number of hidden units of Elman ASRTNs capable of computing arbitrary dichotomies of a set of examples.

5.2.1.1 n:h:1 Lower bound

Theorem 5.12:

For an *n:h:1* Elman ASRTN, there is a set of N input vectors with associated outputs such that we need $\lceil \frac{1}{2}(\sqrt{n^2 + 4(N-1)} - n) \rceil$ hidden threshold units to realize the specified input/output relation, where $h \leq n$.

Proof:

From Theorem 5.1 it follows that an *n:h:1* Elman ASRTN can store at least $N = (n+h)h + 1$

examples in general position. We now obtain an expression for h by rewriting $N = (n+h)h+1$ as a quadratic equation in h,

$$h^2 + nh - N + 1 = 0 \tag{5.22}$$

which has the following positive integer solution

$$h = \lceil \frac{-n \pm \sqrt{n^2 + 4(N-1)}}{2} \rceil \tag{5.23}$$

Since it is required that $h > 0$, $-n/2 < 0$, and $\sqrt{n^2 + 4(N-1)} > 0$, only the positive solution is valid; thus a lower bound for the number of threshold hidden units is $\lceil \frac{1}{2}(\sqrt{n^2 + 4(N-1)} - n) \rceil$. $\triangle$

5.2.1.2 n:h:s Upper bound

Sakurai & Yamasaki (1992) obtained an upper bound of $\lceil (N-s)/n \rceil + s$ for the number of hidden units of an *n:h:s* FFTN. A corresponding upper bound for an *n:h:s* Elman ASRTN is shown next.

Theorem 5.13:

There is a set of N input vectors with associated outputs that can be correctly stored by $\lceil P + \frac{1}{2}(\sqrt{4P^2 + 4nP + 8N + n^2} - n) \rceil$ hidden threshold units of an *n:h:s* Elman ASRTN, where $h \leq n$ and $P(\leq 2^s)$ the different kinds of outputs if $h \geq 2P$.

Proof:

From Theorem 5.3 it follows that $(n+h)(\lfloor \frac{h}{2} \rfloor - P) + P$ is an upper bound of the capacity of *n:h:s* Elman ASRTNs when $h \geq 2P$. We now obtain an expression for h by rewriting the upper bound of capacity and assuming an even h,

$$\begin{aligned} N &= (n+h)((h/2) - P) + P \\ \tfrac{1}{2}h^2 - hP + nh/2 - nP + P - N &= 0 \\ \tfrac{1}{2}h^2 - h(P - n/2) - (nP - P + N) &= 0 \end{aligned} \tag{5.24}$$

which has the following positive integer solution

$$\begin{aligned} h &= \lceil \frac{(2P-n) \pm \sqrt{(2P-n)^2 + 8(nP - P + N)}}{2} \rceil \\ &= \lceil P - n/2 \pm \frac{1}{2}\sqrt{4P^2 + 4nP + 8N + n^2} \rceil \end{aligned} \tag{5.25}$$

Since it is required that $h > 0$, $-n/2 < 0$, and $\sqrt{4P^2 + 4nP + 8N + n^2} > 0$, only the positive solution is valid; thus an upper bound for the number of threshold hidden units for an *n:h:s* Elman ASRTN is $\lceil P + \frac{1}{2}(\sqrt{4P^2 + 4nP + 8N + n^2} - n) \rceil$ if $h \geq 2P$. $\triangle$

5.2.1.3 n:h:s Lower bound

The same lower bound ($\lceil (N-1)/n \rceil$) is obtained for the number of hidden units of feedforward threshold networks having one or s output threshold units [Sakurai & Yamasaki, 1992]. The next theorem shows that this is also the case for *n:h:1* and *n:h:s* Elman ASRTNs.

Theorem 5.14:

For an *n:h:s* Elman ASRTN, there is a set of N input vectors with associated outputs such that we need $\lceil \frac{1}{2}(\sqrt{n^2+4(N-1)}-n) \rceil$ threshold hidden units to realize the specified input/output relation, where $h \leq n$.

Remark:

From Theorem 5.3 it follows that an *n:h:s* Elman ASRTN can store at least $N = (n+h)h+1$ examples in general position, if $P = 2$. Therefore the result follows from that of Theorem 5.13. △

5.2.2 Jordan ASRTNs

5.2.2.1 n:h:1 Upper bound

Cosnard *et al* (1992) showed that an arbitrary dichotomy (S^+, S^-) can be separated by $n(\lfloor \frac{1}{\delta} \rfloor + 1)$ hyperplanes of a feedforward threshold network. In section 5.2 we have showed that this upper bound should be $n(\lfloor \frac{\sqrt{n}}{\delta} \rfloor + 1)$. We next show in a similar fashion an upper bound for the number of hidden units of an *n:h:1* Jordan ASRTN in terms of δ. Note that the input-state space is $[0,1]^{n+1}$ as opposed to $\mathcal{R}^{n+1}$ for this type of bound.

Theorem 5.15:

An arbitrary dichotomy (S^+, S^-) of $[0,1]^{n+1}$ can be separated by $(n+1)(\lfloor \frac{\sqrt{n+1}}{\delta} \rfloor + 1)$ hyperplanes of an *n:h:1* Jordan ASRTN.

Remark:

By applying Cosnard *et al*'s arguments to an *n:h:1* Jordan ASRTN, which are outlined in Appendix D.2.2, an upper bound of $(n+1)(\lfloor \frac{\sqrt{n+1}}{\delta} \rfloor + 1)$ is obtained for the number of hidden units. △

5.2.2.2 n:h:1 Lower bounds

We now first show an lower bound for the number of hidden units in terms of δ and then one in terms of N.

Theorem 5.16:

An arbitrary dichotomy of $[0,1]^{n+1}$ can be realized by an *n:h:1* Jordan ASRTN with at least $\lceil(\frac{(n+1)}{e})\lfloor\sqrt{n+1}/\delta\rfloor\rceil$ hidden threshold units.

Remark:

By following similar steps as in the corrected version of Cosnard *et al*'s proof, which are outlined in Appendix D.2.1, the result follows. $\triangle$

Theorem 5.17:

For an *n:h:1* Jordan ASRTN, there is a set of N input vectors with associated outputs such that we need $\lceil\frac{N-1}{n+1}\rceil$ hidden threshold units to realize the specified input/output relation, where $s \leq n$

Proof:

From Theorem 5.5 it follows that an *n:h:1* Jordan ASRTN can store at least $N = (n+1)h + 1$ examples in general position. We obtain an expression for h by rewriting the equation for the lower bound on the capacity as $h = (N-1)/(n+1)$. Since h is a positive integer, a lower bound for the number of threshold hidden units for an *n:h:1* Jordan ASRTN is $\lceil\frac{(N-1)}{(n+1)}\rceil$. $\triangle$

The two lower bounds proved for *n:h:1* Jordan ASRTNs, being expressed respectively in terms of δ and N, also give complementary results. For a large N and a large δ, $\lceil(\frac{(n+1)}{e})\lfloor\sqrt{n+1}/\delta\rfloor\rceil$ is a better lower bound than $\lceil\frac{N-1}{n+1}\rceil$. When δ is very small the converse is true.

5.2.2.3 n:h:s Upper bound

It was shown by Sakurai & Yamasaki (1992) that $\lceil(N-s)/n\rceil + s$ is an upper bound for the number of hidden units of an *n:h:s* FFTN. The next theorem shows a corresponding upper bound for an *n:h:s* Jordan ASRTN.

Theorem 5.18:

There is a set of N input vectors with associated outputs that can be correctly stored by $\lceil\frac{2P(n+s-1)+N}{(n+s)}\rceil$ hidden threshold units of an *n:h:s* Jordan ASRTN, where $s \leq n$ and $P(\leq 2^s)$ the

different kinds of outputs if $h \geq 2P$.

Proof:

From Theorem 5.8 it follows that $(n+s)(\lfloor \frac{h}{2} \rfloor - P) + P$ is an upper bound of the capacity of *n:h:s* Jordan ASRTNs when $h > 2P$. We now obtain an expression for h by rewriting the upper bound of capacity and assuming an even h,

$$\begin{aligned} N &= (n+s)((h/2) - P) + P \\ h(n+s)/2 &= (n+s)P - P + N \\ h &= 2\frac{P(n+s-1)+N}{n+s} \end{aligned} \tag{5.26}$$

Since h is a positive integer, an upper bound for the number of threshold hidden units for an *n:h:s* Jordan ASRTN is $\lceil \frac{2P(n+s-1)+N}{n+s} \rceil$. $\triangle$

5.2.2.4 n:h:s Lower bound

Theorem 5.19:

For an *n:h:s* Jordan ASRTN, there is a set of N input vectors with associated outputs such that we need $\lceil \frac{(N-1)}{(n+s)} \rceil$ hidden threshold units to realize the specified input/output relation, where $s \leq n$.

Proof:

From Theorem 5.8 it follows that an *n:h:s* Jordan ASRTN can store at least $N = (n+s)h + 1$ examples in general position, if $P = 2$. We obtain an expression for h by rewriting the equation for the lower bound on the capacity as $h = (N-1)/(n+s)$. Since h is a positive integer, a lower bound for the number of threshold hidden units for an *n:h:s* Jordan ASRTN is $\lceil \frac{(N-1)}{(n+s)} \rceil$. $\triangle$

5.2.3 Temporal Autoassociation ASRTNs

In the next two sections an upper and lower bound are obtained for the number of hidden units of an *n:h:s* Temporal Autoassociation ASRTN. The two bounds follow directly from respectively, Theorem 5.13 and Theorem 5.12. The only difference is that $P \leq 2^{n+h+s}$ for an *n:h:s* Temporal Autoassociation ASRTN.

5.2.3.1 n:h:s Upper bound

Theorem 5.20:

There is a set of N input vectors with associated outputs that can be correctly stored by $\lceil P + \frac{1}{2}(\sqrt{4P^2 + 4nP + 8N + n^2} - n)\rceil$ hidden threshold units of an *n:h:s* Temporal Autoassociation ASRTN, where $h \leq n$ and $P(\leq 2^{n+h+s})$ the different kinds of outputs if $h \geq 2P$.

5.2.3.2 n:h:s Lower bound

Theorem 5.21:

For an *n:h:s* Temporal Autoassociation ASRTN, there is a set of N input vectors with associated outputs such that we need $\lceil \frac{1}{2}(\sqrt{n^2 + 4(N-1)} - n)\rceil$ threshold hidden units to realize the specified input/output relation, where $h \leq n$.

5.3 Summary

In this chapter upper and lower bounds were proven for the capacity and number of hidden units of Elman, Jordan, and Temporal Autoassociation ASRTNs. We have addressed the question of how many examples a network can remember not only in terms of the number of examples in general position, but also in terms of the Vapnik-Chervonenkis dimension. The question of how many hidden units are enough to realize an arbitrary function is likewise addressed not only in terms of the number of examples in general position, but also in terms of the distance between two distinct classes.

Apart from proving upper and lower bounds for the capacity of different ASRTNs, we have discovered some mistakes in Sakurai & Yamasaki (1992)'s proof for an upper bound of an *n:h:s* FFTN's capacity and made the appropriate corrections. Table 5.1 summarises upper and lower bounds of the capacity and number of hidden units that we have obtained for *n:h:1* Elman and Jordan ASRTNs. We have also pointed out some mistakes in Cosnard *et al* (1992)'s proofs for determining lower bounds for the number of hidden units of *n:h:1* FFTNs. Table 5.2 contains a summary of the bounds for the capacity and number of hidden units of *n:h:1* FFTNs and corresponds to Table 5.1. Table 5.3 summarises upper and lower bounds of the capacity and number of hidden units of *n:h:s* Elman, Jordan, and Temporal Autoassociation ASRTNs, whereas Table 5.4 contains the corresponding bounds for *n:h:s* FFTNs. We have further compared related bounds

	n:h:1 Elman ASRTNs	*n:h:1* Jordan ASRTNs
Deterministic Capacity		
Lower bound	$(n+h)h+1$	$(n+1)h+1$
Probabilistic Capacity		
Upper bound	$O((n+h+1)h\log_2 h)$	$O((n+2)h\log_2 h)$
VC-dim Capacity		
Upper bound	$2w\log_2(eh)$	$(n+1)h(\log_2 h+2\log_2(\log_2 h))$
Lower bound	$(n+h)h+1$	$(n+1)h+1$
Hidden units		
Upper bound	–	$(n+1)(\lfloor\frac{\sqrt{n+1}}{\delta}\rfloor+1)$
Lower bounds	$\lceil 1/2(\sqrt{n^2+4(N-1)}-n)\rceil$	$\lceil\frac{N-1}{n+1}\rceil$
	–	$\lceil(\frac{(n+1)}{e})(\lfloor\sqrt{n+1}/\delta\rfloor)\rceil$

Table 5.1: Upper and lower bounds for the capacity and number of hidden units of *n:h:1* Elman and Jordan ASRTNs

	n:h:1 FFTNs	References
Deterministic Capacity		
Lower bound	$nh+1$	[Sakurai, 1992]
Probabilistic Capacity		
Upper bound	$O(nh\log_2 h)$	[Mitchison *et al*, 1989]
VC-dim Capacity		
Upper bound	$2w\log_2(eh)$	[Baum & Haussler, 1989]
Lower bound	$nh+1$	[Sakurai, 1993]
Hidden units		
Upper bound	$n(\lfloor\frac{\sqrt{n}}{\delta}\rfloor+1)$	[Cosnard *et al*, 1992]
Lower bounds	$\lceil\frac{N-1}{n}\rceil$	[Sakurai, 1992]
	$\lceil(n/e)\lfloor\sqrt{n}/\delta\rfloor\rceil$	[Cosnard *et al*, 1992]

Table 5.2: Upper and lower bounds for the capacity and number of hidden units of *n:h:1* FFTNs (with references)

	n:h:s Elman ($P \leq 2^s$) & Temporal Auto-association ($P \leq 2^{n+h+s}$) ASRTNs ($h \geq 2P$)	*n:h:s* Jordan ($P \leq ?^s$) ASRTN ($h \geq 2P$)
Deterministic Capacity		
Upper bound	$(n+h)(\lfloor \frac{h}{2} \rfloor - P) + P$	$(n+s)(\lfloor \frac{h}{2} \rfloor - P) + P$
Probabilistic Capacity		
Upper bound	$O((n+h+1)(1+h/s)\log_2(1+h/s)$	$O((n+s+1)(1+h/s)\log_2(1+h/s)$
Hidden units		
Upper bound	$\lceil P + 1/2(\sqrt{4P^2+4nP+8N+n^2}-n)\rceil$	$\lceil \frac{2P(n+s-1)+N}{(n+s)} \rceil$
Lower bound	$\lceil 1/2(\sqrt{n^2+4(N-1)}-n)\rceil$	$\lceil \frac{(N-1)}{(n+s)} \rceil$

Table 5.3: Upper and lower bounds for the capac'ty and number of hidden units of *n:h:s* Elman, Jordan and Temporal Autoassociation ASRTNs

	n:h:s FFTNs ($P \leq 2^s$; $h \geq 2P$)	References
Deterministic Capacity		
Upper bound	$n(\lfloor \frac{h}{2} \rfloor - P) + P$	[Sakurai & Yamasaki, 1992]
Probabilistic Capacity		
Upper bound	$O(n(1+h/s)\log_2(1+h/s)$	[Mitchison *et al*, 1989]
Hidden units		
Upper bound	$2(\lceil \frac{(N-s)}{n} \rceil + s)$	[Sakurai & Yamasaki, 1992]
Lower bound	$\lceil \frac{(N-1)}{n} \rceil$	[Sakurai & Yamasaki, 1992]

Table 5.4: Upper and lower bounds for the capacity and number of hidden units of *n:h:s* FFTNs (with references)

and have given examples in some instances. Finally, all t obtained in this chapter for ASRTNs also apply to architecture-specific recurrent networks with the (continuous) sigmoid activation function when the gain parameter is large.

The next chapter include a summary of the main results obtained in this dissertation and some remarks about further research.

Chapter 6

Summary and Conclusions

This dissertation contains the main results of a pioneering effort to develop novel architectures, training strategies, dynamics analysis techniques and theoretical complexity results for architecture-specific recurrent neural networks (ASRNNs). It is the overall objective of this study to explore ASRNNs in all departments: training, classification, dynamics, and complexity. As ASRNNs are the main focus, we have given some background on recurrent neural networks in general as well as a review of the research literature (Chapter 2). To get a better perspective of ASRNNs in the temporal processing research field, we have constructed a temporal processing framework which described the different approaches taken. We have focused on Elman, Jordan, and Temporal Autoassociation ASRNNs using discrete-time backpropagation and also suggested three new ASRNN architectures.

The typical approach taken by other researchers to address the defects of standard backpropagation is to investigate alternative methods for selecting initial parameter values, or for adjusting parameter values during BP training. In Chapter 3 we have suggested that the training strategy, that is the method for presentation of examples to the network during learning according to some performance criteria, is a viable alternative method to produce an effective solution within a feasible time. The dual purpose was to evaluate training strategies and ASRNNs simultaneously for different applications, which vary in complexity and range from sequence recognition to sequence generation. It was demonstrated with six different ASRNNs and feedforward networks that for several applications, *Increased Complexity Training* (ICT) outperformed *Combined Subset Training* (CST) and *Fixed Set Training* (FST). We have also compared several ASRNNs for the Counting and Addition tasks, and found the Output-to-Hidden Hidden-to-Hidden ASRNN, proposed in Chapter 2, to be the superior architecture, outperforming the conventional ASRNNs by between 44% and 87% for the different training strategies. The best features of the Elman

and Jordan ASRNN are combined within this network, causing it to be able to learn input and/or output sequences. Since the Elman, TA, and Output-to-Hidden Hidden-to-Hidden ASRNNs have a context layer on the input level, it is suited to effectively deal with input sequences as was experimentally shown. The Jordan ASRNN is restricted in not being able to remember input not reflected in its output, whereas the TA ASRNN can only deal with tasks that allow learning an inverse mapping of the current input and context units. The Output-to-Output Hidden-to-Hidden and Output-to-Output ASRNN, both having output-to-output feedback, did not prove to be beneficial for tasks with input sequences.

For both ICT and CST the method of determining the required RMS termination criteria per subset was unsatisfactory since it required experimentation in stead of being performed algorithmically. In answer to the latter need, we have proposed incremental training strategies, *Incremental Subset Training* (IST) and *Incremental Increased Complexity Training* (IICT), which improved the convergence rate compared to CST, FST, and even ICT. IST and IICT have showed for Addition that a good training set can be quite small to provide very good generalization. IST, for example, for a particular simulation needed only half of the training set to double the performance of training on a fixed set. It also improved performance by 50% compared to FST on the same half of the training set. For the correct increment in subset size the number of updates required for Addition almost attained the theoretical lower bound. The incremental strategies also suggest a schedule for RMS termination values on each subsequent subset. The proposed incremental training strategies reduce the effect of many attractor basins which cause the erratic behaviour of the error curve, thus leading to faster convergence.

We have also proposed six Delta training strategies by first employing the Delta Ranking Method, which determine the complexity relation between the input patterns by obtaining their inter-pattern distances and then ranking them according to some scheme. We have introduced three basic ranking schemes which led to *Smallest Delta Subset Training* (SDST), *Largest Delta Subset Training* (LDST), *Alternating Delta Subset Training* (ADST), their incremental versions, and also their epoch versions. All the Delta training strategies proved to be very effective (evaluated with different applications) in reducing the training time when compared to the conventional strategies. Incremental Delta training strategies performed the best overall, signalling that the ordering of training patterns according to our proposed delta ranking schemes, especially when presented in an incremental fashion, forces the network to discriminate between classes early in the training process, leading to reduced training time. By visualizing the movement of hyperplanes in the input space and the movement of particular weights in the weight space for each training strategy, it was shown how the new training strategies efficiently reduce the weight

activity and obtain faster convergence. The training strategies should be regarded as different tools in a training strategy toolbox, each one suited for its particular purpose. For example, IST is suited for applications where the fixed set of input patterns can be partitioned into subsets of increasing complexity, whereas SDST is more suited for tasks where the inter-pattern distances of the input patterns can be easily obtained.

In Chapter 4 the classification capabilities and dynamics of the three main architecture-specific recurrent networks, namely Elman, Jordan, and Temporal Autoassociation networks were investigated. For the theoretical analysis, we have considered the threshold version of these networks. It was pointed out that an architecture-specific recurrent threshold network can be viewed as a feedforward threshold network with special additional inputs, the context or state units, whose values are provided by the network itself. For each network the possible types and number of cells in the input and hidden unit activation space were determined. For example, it was shown that for an Elman ASRTN there can be no closed or imaginary cells and all of them (2^h) are open. It was also concluded that Elman ASRTNs are not capable of forming disconnected regions. These capability results also hold for Jordan ASRTNs when the number of hidden units is smaller or equal to the sum of the number of input and state units. Account was given of the effect of adding context and state units by comparing *n:h:s* FFTNs with their corresponding *n:h:s* ASRTNs in terms of various examples and interpreting the equations for the number of cells. It was also pointed out that the conclusions of this analysis can be extended to networks with the well-known sigmoid activation function when the gain parameter is large.

The dynamics of the classification process was further explored by analyzing the clusters formed by the hidden unit activations of the network. It was shown with four different applications that an Elman ASRNN can learn to simulate a finite state machine, which was derived from these clusters. The internal representation of the network were obtained by visualizing the input activation space and using clustering techniques, such as Hierarchical Cluster Analysis, Principal Component Analysis, and Sammon Transformation Analysis. The latter was shown to be a superior clustering technique when compared to PCA. The correspondence between the simulated FSM and the one initially constructed for the Addition training data was also determined. For Addition the construction of the Mealy machine also enabled the identification of non-deterministic elements in the training data. The application of five training strategies demonstrated that training is much easier (all of them between 34% and 44%) with deterministic data as opposed to non-deterministic data, even though the difference in the two training sets was only 0.6%. IST, the best training strategy, improved performance in the non-deterministic case by 24% compared to fixed set training and in the deterministic case by 19% on average. The

dynamics of the learning process in ASRNNs were then explored by visualizing the evolution of clusters formed by Sammon Transformation Analysis during the learning process. It was also discussed in terms of a mapping of inspection points on the learning curve, simulation results at each point, and learning curve behaviour. This furthered our understanding of how the network learns to distinguish between different input sequences and how clusters are formed. We have observed the network's learning difficulties and how it eventually overcomes these difficulties by visualizing the hidden activation time traces during learning. It was also shown that the network's performance improves with longer hidden unit activation histories. The capability of Jordan and Elman networks was further compared. We have argued that the former's inability to deal with tasks where the input is not reflected directly in the output, cannot be related to the number or the types of cells. When the number of hidden units is greater than the sum of the number of input and the number of state units, an Elman ASRTN has always more cells in the input space than a Jordan ASRTN. It was finally pointed out that the capability results obtained for the Elman network also hold for the Temporal Autoassociation network.

In Chapter 5 the complexity analysis of Elman, Jordan, and Temporal Autoassociation ASRTNs was continued by proving upper and lower bounds for their capacity and number of hidden units. We have addressed the capacity of the network in terms of the number of examples in general position and the Vapnik-Chervonenkis dimension. The question of how many hidden units are enough to realize an arbitrary function is addressed in terms of the number of examples in general position and the distance between two distinct classes. Since the number of hidden units is in practice chosen experimentally, bounds for the number of hidden units is of particular importance for the optimal design of an ASRTN architecture. For Elman and TA ASRTNs the number of context units is also dependent on the number of hidden units. We have not only proven upper and lower bounds for *n:h:1* and *n:h:s* ASRTNs, but also gave some examples and compared related bounds. An *n:h:1* Elman ASRTN, for example, has a larger capacity lower bound, $(n+h)h+1$ examples, when compared to an *n:h:1* Jordan ASRTN's lower bound of $(n+1)h+1$. Their respective lower bounds for the number of hidden units are $\lceil 1/2(\sqrt{n^2+4(N-1)}-n)\rceil$ and $\lceil \frac{N-1}{n+1} \rceil$. For *n:h:s* Elman and Temporal Autoassociation ASRTNs, the same bounds are obtained, except that the former has the precondition that $P \leq 2^s$ and the latter that $P \leq s^{n+h+s}$. Examples of capacity upper bounds for *n:h:s* Elman and Jordan ASRTNs are respectively, $(n+h)(\lfloor \frac{h}{2} \rfloor - P) + P$ and $(n+s)(\lfloor \frac{h}{2} \rfloor - P) + P$. Apart from proving ASRTN bounds, we have discovered some mistakes in FFTN proofs and made the appropriate corrections. For example, after correcting mistakes in Sakurai & Yamasaki (1992)'s proof for an upper bound of an *n:h:s*

FFTN's capacity, it was determined that the actual upper bound should be

$$n(\lfloor\frac{h}{2}\rfloor - (P-2) + \lceil(\lfloor\frac{h}{2}\rfloor + 1)/(P-1)\rceil - 1) + (P-1) > n(\lfloor\frac{h}{2}\rfloor - P) + P$$

Again the conclusions of this complexity analysis can be extended to architecture-specific recurrent neural networks with the sigmoid activation function. Applying the theorems and results in this study to experiments should point to more optimal design of ASRNN architectures.

Currently the field of ASRNNs is experiencing rapid development due to their applications, simplicity, and powerful ability to process and generate temporal sequences. The most recent literature on ASRNNs include cases where ASRNNs are combined with non-recurrent approaches such as RAAM [Pollack, 1990] and time-delay networks. This hybrid approach suggests the way that novel neural networks in the temporal processing field should be developed. The lack of self-organizing or even biological plausible learning algorithms for ASRNNs should also be addressed.

Although the training strategies for ASRNNs proved to be very effective, certain aspects of these strategies can be further researched. The incremental training strategies include an incremental schedule that algorithmically determines the RMS termination values on each subsequent subset for any subset increment. Since the determination of the optimum subset increment still requires experimentation, a method that automatically deals with this problem should be developed. For IICT we have investigated the distribution of temporal patterns from each complexity class. It was conjectured that the distribution of temporal patterns from each complexity class should not assign too small importance to the less complex classes. However, exactly how the temporal patterns should be distributed must be further investigated. The training time of ASRNNs can be further improved when the training strategies are used together with adaptive learning and momentum rates. For example, for ICT the learning rate may be increased when training patterns from a more complex subset are presented. Since the training strategies proved to be very successful in reducing the training time for ASRNNs and feedforward networks, it is conjectured that the strategies should also be very effective for general-purpose recurrent networks as well as non-recurrent temporal approaches.

The classification capabilities of Elman, Jordan, and Temporal Autoassociation ASRNNs were examined in terms of the number and possible types of cells the network is capable of forming in the input and hidden activation spaces during classification. This investigation should also be extended to include other ASRNNs, especially the promising Output-to-Hidden Hidden-to-Hidden ASRNN (proposed in Chapter 2). The clusters formed by the hidden unit activation values of the Jordan and Temporal Autoassociation networks can also be analyzed. A compar-

ison of the clusters formed by different ASRNNs should give insight in how their distributed representations are related. It was shown that an Elman ASRNN can learn to simulate a finite state machine. Research should also be conducted to determine the correspondence between the finite state machines that different ASRNNs learn to simulate. The learning dynamics of Jordan and Temporal Autoassociation can also be explored by visualizing how their internal representations evolve over time during training. Such an analysis should further ones understanding of how the networks form clusters and learn to distinguish between different input sequences. Further work should also focus on analyzing the dynamics of the learning process in terms of trajectories formed between states and the type of attractors developed. The capability analysis can be extended to link the possible types and number of cells in the input and hidden unit activation space with the clusters formed during the learning process.

The capacity of ASRNNs can also be investigated with finite state machines, since their memory requirements may be precisely controlled and they are relatively well understood. It should be interesting to see if different clusters are formed for a varying number of hidden units. In particular, are the clusters a minimum when the network has a minimal number of hidden units? The complexity analysis of ASRNNs should further be developed to improve upper and lower bounds for the capacity and number of hidden units. We have obtained theoretical results for Elman, Jordan, and Temporal Autoassociation ASRNNs. This analysis should be extended to include other ASRNNs, such as the Output-to-Hidden Hidden-to-Hidden ASRNN.

In this dissertation the foundations have been laid for the effective use of the promising techniques developed and results obtained for architectures, training strategies, dynamics analysis, and complexity analysis of architecture-specific recurrent neural networks.

Bibliography

[Abramowitz *et al*, 1972] Abramowitz, M., Stegun, I.A., *Handbook of Mathematical Functions*, Dover, New York, 1972.

[Almeida, 1987] Almeida, L.B., "A Learning Rule for Asynchronous Perceptrons with Feedback in Combinatorial Environment", *1st IEEE International Conference on Neural Networks*, pp. II:609-618, 1987.

[Amari, 1972] Amari, S., "Learning Patterns and Pattern Sequences by Self-Organizing Nets of Threshold Elements", *IEEE Transactions on Computers*, C-21, pp. 1197-1206, 1972.

[Anderson, 1968] Anderson, J., "A Memory Storage Model Utilizing Spatial Correlation Functions", *Kybernetik*, Vol. 5, pp. 113-119, 1968.

[Anderson, 1977] Anderson, J., "Neural Models with Cognitive Implications", *Basic Processes in Reading Perception and Comprehension*, eds. D. Laberge and S. Samuels, Hillsdale, New Jersey: Lawrence Erlbaum Associates, pp. 413-451, 1977.

[Anderson *et al*, 1988] Anderson, S., Merrill, J., & Port, R., "Dynamic Speech Categorization with Recurrent Networks", Technical Report 258, Department of Linguistics and Computer Science, Indiana University, 1988.

[Bachrach, 1988] Bachrach, J., *Learning to represent state*, Unpublished master's thesis, University of Massachusetts, Amherst, 1988.

[Barto *et al*, 1983] Barto, A., Sutton, R., & Anderson, C., "Neuron-like Adaptive Elements that can Solve Diffifcult Learning Control Problems", *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13, pp. 834-846, 1983.

[Barto & Anandan, 1985] Barto, A., & Anandan, P., "Pattern Recognizing Stochastic Learning Automata", *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15, pp. 360-375, 1985.

[Banzhaf, 1991a] Banzhaf, W., "Processing Spatio-Temporal Patterns by Mapping Time into Intensity" *Proceedings of the IJCNN'91*, Seattle, pp. II: 871-877, 1991.

[Banzhaf & Kyuma, 1991b] Banzhaf, W., & Kyuma, K., "The Time-into-Intensity-Mapping Network" *Biological Cybernetics*, Vol. 66, pp. 115-121, 1991.

[Banzhaf & Kyuma, 1992] Banzhaf, W., & Kyuma, K., "Pattern Segmentation using the TIM Neural Network Architecture" *2nd Int. Conference on Fuzzy Systems and Neural Networks*, IIZUKA-92, 1992.

[Bartell & Cottrell, 1991] Bartell, B.T., & Cottrell, G.W., "A Model of Symbol Grounding in a Temporal Environment", *Proceedings of the IJCNN'91*, Seattle, pp. I: 805-810, 1991.

[Baum, 1988] Baum, E.B., "On the Capabilities of Multilayer Perceptrons", *Journal of Complexity*, Vol. 4, pp. 193-215, 1988.

[Baum & Hausler, 1989] Baum, E.B., & Haussler, D., "What Size Net Gives Valid Generalization", *Neural Computation*, Vol. 1, pp. 151-160, 1989.

[Bianchini *et al*, 1994] Bianchini, M., Gori, M., & Maggini, M., "On the Problem of Local Minima in Recurrent Neural Networks", *IEEE Transactions on Neural Networks*, Vol. 5, No. 2, pp. 167-177, March 1994.

[Bourlard & Wellekens, 1988] Bourlard, H., & Wellekens, C.J., "Links Between Markov Models and Multilayer Perceptrons", Technical Report Manuscript M-263, Phillips Research Laboratory, Brussels, Belgium, 1988.

[Buhmann & Schulten, 1988] Buhmann, J., & Schulten, K., "Storing Sequences of Biased Patterns in Neural Networks with Stochastic Dynamics" *Neural Computers*, eds. Eckmiller, R., v.d. Malsburg, C., Springer-verlag, New York, pp. 231-242, 1988.

[Carpenter & Grossberg, 1986] Carpenter, G.A., & Grossberg, S., "Adaptive Resonance Theory: Stable Self-Organization of Neural Recognition Codes in Response to Arbitrary Lists of Input Patterns", *8th Annual Conference of the Cognitive Science Society*, Hillsdale, NJ: Lawrence Erlbaum Associates, pp. 45-62, 1986.

[Cloete & Ludik, 1993] Cloete, I., & Ludik, J., "Increased Complexity Training", in *New Trends in Neural Computation*, eds. Mira, J., Cabestany, J., Prieto, A., ... s Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 267-271, 1993.

[Cloete & Ludik, 1994a] Cloete, I., & Ludik, J., "Incremental Training Strategies", *Proceedings of the International Conference on Artificial Neural Networks (ICANN'94)*, Sorrento, Italy, Vol II, pp. 743-746, 26-29 May 1994.

[Cloete & Ludik, 1994b] Cloete, I., & Ludik, J., "Delta Training Strategies", *Proceedings of the IEEE World Congress on Computational Intelligence (WCCI'94)*, Orlando, Florida, USA, Vol I, pp. 295-298, 26 June to 2 July 1994.

[Cleeremans *et al*, 1989] Cleeremans, A., Servan-Schreiber, D., & McClelland, J.L., "Finite State Automata and Simple Recurrent Networks", *Neural Computation*, Vol. 1, pp. 372-381, 1989.

[Cohen & Grossberg, 1983] Cohen, M. & Grossberg, S., "Absolute Stability of Global Pattern Formation and Parallel Memory Storage by Competitive Neural Networks", *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13, pp. 815-825, 1983.

[Cosnard *et al*, 1992] Cosnard, M., Koiran, P., & Paugam-Moisy, H., "Complexity Issues in Neural Network Computations", *Proceedings of the LATIN'92*, Sao Paulo, Brazil, April 6-10, 1992.

[Cottrell & Tsung, 1991] Cottrell, G.W., & Tsung, F.S., "Learning Simple Arithmetic Procedures", *High-Level Connectionist Models*, eds. J.A. Barnden, J.B. Pollack, in the series Advances in Connectionist and Neural Computation Theory, Vol. 1, pp.305-321, 1991.

[Cover, 1965] Cover, T.M., "Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition", *IEEE Transactions on Electronic Computers*, Vol. 14, pp. 326-334, 1965.

[Debar & Dorizzi, 1992] Debar, H., & Dorizzi, B., "An Application of a Recurrent Network to an Intrusion Detection System", *Proceedings of the IJCNN'92*, Baltimore, MD, USA, Vol. II, pp. 478-483, June 1992.

[Demura *et al*, 1993] Demura, K., Kajiura, M., & Anzai, Y., "Dynamic Recurrent Neural Networks for Real Time Learning" *Proceedings of the IJCNN'93*, Nagoya, Japan, Vol. III, pp. 2646-2649, October 1993.

[Diederich, 1992] Diederich, J., "Neural Networks for Electronic Communication", Colloquium Notes, German National Research Center for Computer Science, St. Augustin, Germany, 1992.

[Doya & Yoshizawa, 1989] Doya, K., & Yoshizawa, S., "Adaptive Neural Oscillator using Continuous-Time Back-Propagation Learning", *Neural Networks*, Vol. 2, pp. 375-385, 1989.

[Doya, 1992] Doya, K., "Bifurcations in the Learning of Recurrent Neural Networks", Preprint, Department of Biology, University of California, San Diego, USA, 1992.

[Edelman, 1988] Edelman, G.M., *Neural Darwinism: The Theory of Neuronal Group Selection*, Basic Books, New York, 1988.

[Elman *et al*, 1986] Elman, J.L., & McClelland, J.L., "Exploiting Lawful Variability in the Speech Wave", in *Invariance and Variability in Speech Processes*, eds. J.S. Perkell and D.H. Klatt, New Jersey: Lawrence Erlbaum, 1986.

[Elman, 1988] Elman, J.L., "Finding Structure in Time", CRL Technical Report 8801, University of California, San Diego, 1988.

[Elman, 1989] Elman, J.L., "Structured Representations and Connectionist Models", CRL Technical Report 8901, University of California, San Diego, 1989.

[Elman, 1990] Elman, J.L. "Finding Structure in Time", *Cognitive Science*, Vol. 14, pp. 179-211, 1990.

[Elman, 1991] Elman, J.L., "Incremental Learning, or the Importance of starting small", CRL Technical Report 9101, Center for Research in Language University of California, San Diego, March 1991.

[Fahlman, 1989] Fahlman, S.E., "Faster Learning Variations on Back-propagation: An Empirical Study", *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann Publishers, pp. 38-50, 1989.

[Fanelli, 1994] Fanelli, R., "Asymptotically Stable Automaton-like Behaviour in Recurrent Neural Networks", *Proceedings of the WCNN'94*, San Diego, USA, Vol. III, pp. 268-274, June 1994.

[Fernando *et al*, 1992] Fernando, S., Islam, F., Utama, P., & Watson K., "High Impedance Fault Detection using Recurrent Networks", *Artificial Neural Networks*, eds. Alexander I., & Taylor, J., Elsevier Science Publishers B.V., Vol. 2, pp. 1615-1618, 1992.

[Fiesler & Caulfield, 1992] Fiesler, E., and Caulfield, H.J., "Neural Network Formalization", Preprint, IDIAP, Case postale 609, CH-1920 Martigny, Suisse, 1992.

[Ghahramani & Allen, 1991] Ghahramani, Z., & Allen, R.B., "Temporal Processing with Connectionist Networks", *Proceeding of the IJCNN'91*, Seattle, Washington, USA Vol. 2, pp. 541-546, 1991.

[Giles *et al*, 1992] Giles, C.L., Miller, C.B., Chen, D., Chen, H.H., Sun, G.Z., & Lee, Y.C., "Learning and extracting finite state automata with second-order recurrent neural networks", *Neural Computation*, Vol. 4, pp. 393-405, 1992.

[Goldberg, 1989] Goldberg, D.E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA, USA, 1989.

[Gordon *et al*, 1991] Gordon, A., Steele, J.P.H., & Rossmiller, K., *Intelligent Engineering Systems through Artificial Neural Networks*, eds. Dagli, C.H., Kumara, S.R.T., Shin, Y.C., ASME Press, ANNIE-91, St. Louis, Missourri, pp. 365-370, November 1991.

[Grossberg, 1968] Grossberg, S., "Some Nonlinear Networks Capable of Learning a Spatial Pattern of Arbitrary Complexity", *Proceedings of the National Academy of Science*, Vol. 59, pp. 368-372, 1968.

[Grossberg, 1973] Grossberg, S., "Contour Enhancement, Short-Term Memory, and Constancies in Reverberating Networks", *Studies in Applied Mathematics*, Vol. 52, pp. 217-257, 1973.

[Hebb, 1949] Hebb, D.O., *The Organization of Behavior: A Neuropsychological Theory*, John Wiley and Sons, New York, 1949.

[Hester *et al*, 1994] Hester, K.A., Bringmann, M.J., Langan, D., Niccolai, M.J., & Nowack, W.J., "The Predictive RAAM: A RAAM That Can Learn to Distinguish Sequences from a Continuous Input Stream", *Proceedings of the WCNN'94*, San Diego, USA, Vol. IV, pp. 97-103, June 1994.

[Hindmarsh & Rose, 1984] Hindmarsh, J.L., & Rose, R.M., "A Model of Neuronal Bursting Using Three Coupled First Order Differential Equations", *Proceedings of the Royal Society of London, B*, Vol. 221, pp. 87-102, 1984.

[Hinton *et al*, 1984] Hinton, G., Ackley, D., & Sejnowski, T., "Boltzmann machines: Constraint Satisfaction Networks that Learn", Technical Report CMU-CS-84-119, Department of Computer Science, Carnegie Melton University, 1984.

[Hodgkin & Huxley, 1952] Hodgkin, A.L., & Huxley, A.F., "A Quantitive Description of Membrane Current and its Application to Conduction end Exitation in a Nerve", *Journal of Physiology*, London, Vol. 117, pp. 500-544, 1952.

[Hoekstra & Kooijman, 1991] Hoekstra, J., Kooijman, R., "Recurrence with Delayed Links in Multilayer Networks for Processing Sequential Data", *Artificial Neural Networks*, eds. Kohonen, T., Mäkisara, [illegible] Simula, O., Kangas, J., Elsevier Science Publishers B.V. (North-Holland), pp. 1149-1152, 1991.

[Hoekstra, 1992] Hoekstra, J., "Is Counting with Artificial Neural Networks a Problem?", *Proceedings of the 1st IFIP Working Group-10.6 Workshop*, INPG, Grenoble, France, pp. 27-30, 2-3 March 1992.

[Hopfield, 1982] Hopfield, J., "Neural Networks and Physical Systems with Emergent Collective Properties", *Proceedings of the National Academy of Sciences*, USA, Vol. 79, pp. 2554-2558, 1982.

[Hopfield, 1984] Hopfield, J., "Neurons with Graded Response have Collective Computational Properties like those of Two-State Neurons", *Proceedings of the National Academy of Sciences*, USA, Vol. 81, pp. 3088-3092, 1984.

[Imai *et al*, 1992] Imai, K., Gouhara, K., & Uchikawa, Y., "Recognition of Printed Sequential Plural Patterns Using the 3-Layered BP Model with Feedback Connections", *Proceedings of the IJCNN'92*, Baltimore, MD, USA, Vol. III, pp. 754-759, June 1992.

[Iooss, 1991] Iooss, C., "From Lattices of Phonemes to Sentences: A Recurrent Neural Network", *Proceedings of the IJCNN'91*, Seattle, pp. II: 833-838, 1991.

[Jain, 1991] Jain, A.N., "Parsing Complex Sentences with Structured Connectionist Networks", *Neural Computation*, Vol. 3, pp. 110-120, 1991.

[Jordan, 1986] Jordan, M.I., "Attractor Dynamics and Parallelism in a Connectionist Sequential Machine", *Proceedings of the 8th Conference on Cognitive Science*, Amherst, MA, USA, pp. 531-546, 1986.

[Judd, 1988] Judd, S., "Learning in Networks is Hard", *Journal of Complexity*, Vol. 4, pp. 177-192, 1988.

[Karjala *et al*, 1992] Karjala, T.W., Himmelblau, D.M., & Miikkulainen, R., "Data Rectification using Recurrent (Elman) Neural Networks", *Proceedings of the IJCNN'92*, Baltimore, MD, USA, Vol. II, pp. 901-906, June 1992.

[Kleene, 1956] Kleene, S.C, "Representation of Events in Nerve Nets and Finite Automata", In *Automata Studies*, eds., Shannon C.E. & McCarthy J., pp.3-42, Princeton University Press, Princeton, N.J., USA, 1956.

[Kleinfield, 1986] Kleinfield, D., "Sequential State Generation by Model Neural Networks", *Proceedings National Academy Science, Biophysics*, USA, Vol. 83, pp. 9469-9473, 1986.

[Klopf, 1987] Klopf, A., "Drive-Reinforcement Learning: A Real-Time Learning Mechanism for Unsupervised Learning", *Proceedings of the IEEE First ICNN*, Vol. II, San Diego, pp. 441-446, 1987.

[Kobayashi & Hara, 1993] Kobayashi, H., & Hara, F., "Dynamic Recognition of Basic Facial Expressions by Discrete-time Recurrent NNeural Network", *Proceedings of the IJCNN'93*, Nagoya, Japan, Vol. I, pp. 155-158, October 1993.

[Kohonen, 1988] Kohonen, T., "An Introduction to Neural Computing", *Neural Networks*, Vol. 1, pp. 3-16, 1988.

[Kolen & Pollack, 1991] Kolen, J.F., & Pollack, J.B., "Backpropagation is sensitive to initial conditions", *NIPS3*, eds. R.P. Lippmann, J.E. Moody, D.S. Touretzky, pp. 860-886, 1991.

[Kosko, 1986] Kosko, B., "Fuzzy Cognitive Maps", *International Journal of Man-Machine Studies*, Vol. 24, pp. 65-75, 1986.

[Kosko, 1987a] Kosko, B., "Adaptive Bidirectional Associative Memories", *Applied Optics*, Vol. 26, pp 4947-4960, 1987.

[Kosko, 1988a] Kosko, B., "Bidirectional Associative Memories", *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-18, pp. 42-60, 1988.

[Kosko, 1988b] Kosko, B., "Feedback Stability and Unsupervised Learning", *Proceedings of the IEEE ICNN'88*, Vol. I, San Diego: IEEE, pp. 141-152, 1988.

[Lang & Hinton, 1988] Lang, K.J., & Hinton, G.E., "The Development of the Time-Delay Neural Network Architecture for Speech Recognition", Technical Report CMU-CS-88-152, Carnegie-Mellon University, 1988.

[Leighton & Conrath, 1991] Leighton, R.R., & Conrath, B.C., "The Autoregressive Backpropagation Algorithm", *Proceedings of the IJCNN'91*, Seattle, pp. II: 369-377, 1991.

[Linsker, 1988] , "Self-Organization in a Perceptual Network", *IEEE Computer*, Special Issue ... l Neural Systems, Vol. 21, Nr. 3, pp. 105-117, 1988.

[Lipschutz, 1968] Lipshutz, S., "Basis and Dimension", *Schaum's outline of Theory and Problems of Linear Algebra*, McGraw-Hill Book Company, New York, 1968.

[Ludik, 1992] Ludik, J., "Temporal Processing in Neural Networks", Technical Report, Department of Computer Science, University of Stellenbosch, August 1992.

[Ludik & Cloete, 1993] Ludik, J., & Cloete, I., "Training Schedules for Improved Convergence", *Proceedings of the International Joint Conference of Neural Networks (IJCNN'93)*, Nagoya, Japan Vol. I, pp. 561-564, October 1993.

[Ludik *et al*, 1994] Ludik, J., Van der Poel, E., & Cloete, I., "Identification of Finite State Automata in Simple Recurrent Networks", *Proceedings of the World Congress on Neural Networks (WCNN'94)*, San Diego, California, USA, Vol. III, pp. 708-713, 4-9 June 1994.

[Ludik & Cloete, 1994] Ludik, J., & Cloete, I., "Incremental Increased Complexity Training", *Proceedings of the European Symposium on Artificial Neural Networks (ESANN'94)*, Brussels, Belgium, pp. 161-165, 20-21 April 1994.

[MacKay, 1987] MacKay, D.G., *The Organization of Perception and Action*, New York: Springer Verlag, 1987.

[Makhoul *et al*, 1989] Makhoul, J., Schwartz, R., & El-Jaroudi, A., "Classification Capabilities of Two-layer Neural Nets", *IEEE Transactions on Neural Networks*, pp. 635-638, 1989.

[Marcus *et al*, 1991] Marcus, C.M., & Westervelt, R.M., "Dynamics of Analog Neural Networks with Time Delay", *Intelligent Engineering Systems through Artificial Neural Networks*, eds. Dagli, C.H., Kumara, S.R.T., Shin, Y.C., ASME Press, ANNIE-91, St. Louis, Missourri, pp. 568-576, November 1991.

[Marslen-Wilson, 1987] Marslen-Wilson, W.D., "Functional Parallelism in Spoken Word-Recognition", *Spoken Word Recognition*, eds. U.H. Frauenfelder and L.K. Tyler, Cambridge, MA: MIT Press, 1987.

[McCann & Kalman, 1994] McCann, P.J., & Kalman, B.L., "Batch Parallel Training of Simple Recurrent Neural Networks" *Proceedings of the WCNN'94*, San Diego, USA, Vol. III, pp. 533-538, June 1994.

[McCullogh & Pitts, 1943] McCullogh, W., & Pitts, W., "A Logical Calculus of the Ideas Immanent in Nervous Activity", *Bulletin of Mathematical Biophysics*, Vol. 7, pp. 115-133, 1943.

[McDermott & Katagari, 1988] McDermott, E., & Katagari, S., "Phoneme Recognition Using Kohonen's Learning Vector Quantization",*ATR Workshop on Neural Networks and Parallel Distriduted Processing*, Osaka, Japan, 1988.

[Minsky, 1967] Minsky, M.L., in *Computation: Finite and Infinite Machines*, Chapter 3: "Neural Networks. Automata Made Up of Parts", Prentice-Hall Inc., Englewood Cliffs, New Jersey, USA, 1967.

[Mitchison *et al*, 1989] Mitchison, G.J., Durbin, R.M., "Bounds on the Learning Capacity of Some Multi-Layer Networks", *Biological Cybernetics*, Vol. 60, pp. 345-356, 1989.

[Mozer, 1988] Mozer, M.C., "A focused backpropagation algorithm for temporal pattern recognition", Technical Report, University of Toronto, Departments of Psychology and Computer Science, 1988.

[Pearlmutter, 1989] Pearlmutter, B.A., "Learning State Space Trajectors in Recurrent Neural Networks", *Neural Computation*, Vol. 1, pp. 263-269, 1989.

[Pineda, 1987] Pineda, F.J., "Generalization of Back-Propagation to Recurrent Neural Networks" *Physical review letters*, Vol. 59, pp. 2229-2232, 1987.

[Pineda, 1988] Pineda, F.J., "Generalization of Backpropagation to Recurrent and Higher Order Neural Networks", *Advances in Neural Information Processing Systems*, ed. D. Anderson, AIP Conference Proceedings, pp. 602-611, 1988.

[Pollack, 1990] Pollack, J.B., "Recursive Distributed Representations", *Artificial Intelligence*, Vol. 46, pp. 77-105, 1990.

[Port & Anderson, 1989] Port, R.F., & Anderson, S., "Recognition of Melody Fragments in Continuously Performed Music", *Proceedings of the Eleventh Annual Meeting of the Cognitive Science Society*, eds. G. Olson and E. Smith, Hillsdale, NJ: Erlbaum, pp. 820-827, 1989.

[Prager *et al*, 1986] Prager, R.W., Harrison, T.D., & Fallside, F., "Boltzmann Machines for Speech Recognition" *Computer Speech and Language*, Vol. 1, pp.2-27, 1986.

[Psarrou & Buxton, 1994] Psarrou, A., & Buxton, H., "Motion Analysis with Recurrent Neural Nets" *Proceedings of the WCNN'94*, San Diego, USA, Vol. I, pp. 664-667, June 1994.

[Reiss & Taylor, 1991] Reiss, M., & Taylor, J.G., "Storing Temporal Sequences", *Neural Networks*, Vol. 4, No. 6, pp. 773-787, 1991.

[Robinson & Fallside, 1988a] Robinson, A.J., & Fallside F., "A Dynamic Connectionist Model for Phoneme Recognition", *NEURO 1988*, Paris, France, 1988.

[Rohwer & Forest, 1987] Rohwer, R., & Forest, B., "Training Time-Dependencies in Neural Networks" *Proceedings of the 1st IEEE ICNN*, pp. II: 701-708, 1987.

[Rohwer, 1989] Rohwer, R., "The 'Moving Targets' Training Algorithm", *Proceedings of the DANIP*, eds. J. Kinderman and A. Linden, Bonn, GMD, 1989.

[Rosenblatt, 1957] Rosenblatt, F., "The Perceptron: A Perceiving and Recognizing Automaton (project PARA)", Cornell Aeronautical Laboratory Report, 85-460-1, 1957.

[Rumelhart *et al*, 1986a] Rumelhart, D.E., Hinton G.E., Williams R.J., "Interactive Processes in Speech Perception: The TRACE Model", *Parallel Distributed Processing: Vol. 2, Psychological and Biological Models*, eds. Rumelhart, D.E., McClelland, J.L., Cambridge, MA: MIT Press, 1986.

[Rumelhart *et al*, 1986b] Rumelhart, D.E., Hinton G.E., & Williams R.J., "Learning Internal Representations by Error Propagation", *Parallel Distributed Processing: Vol. 1, Foundations*, eds. Rumelhart, D.E., McClelland, J.L., Cambridge, MA: MIT Press, 1986.

[Ryeu & Tak, 1993] Ryeu, J.K., & Tak, H.Y., "Recognition of Korean Spoken Digit Using Single Layer Recurrent Neural Networks", *Proceedings of the IJCNN'93*, Nagoya, Japan, Vol. I, pp. 267-270, October 1993.

[Sakurai, 1992] Sakurai, A., "n-h-1 Networks Store No Less nh+1 Examples, but Sometimes No More", *Proceedings of the WCNN'92*, Vol. III, pp. 936-941, 1992.

[Sakurai & Yamasaki, 1992] Sakurai A., & Yamasaki, M., "On the Capacity of n-h-s Networks", *Artificial Neural Networks*, Vol. 2, Editors: Aleksander, I., Taylor, J., Elsevier Science Publishers B.V., pp. 237-240, 1992.

[Sakurai, 1993] Sakurai, A., "Tighter Bounds of the VC-Dimension of Three-layer Networks", *Proceedings of the WCNN'93*, Vol. III, pp. 540-543, 1993.

[Sammon, 1969] Sammon, J.W. Jr., "A Nonlinear mapping for Data Structure Analsysis", *IEEE Transactions on Computers*, C-18(5):401-409, May 1969.

[Schmidhuber, 1992b] Schmidhuber, J., "A Fixed Size Storage $O(n^3)$ Time Complexity Learning Algorithm for Fully Recurrent Continually Running Networks", *Neural Computation*, Vol. 4, pp. 243-248, 1992.

[Scholtes, 1991a] Scholtes, J.C., "Unsupervised Context Learning in Natural Language Processing", *Proceedings of the IJCNN'91*, Seattle, pp. I: 107-112, 1991.

[Scholtes, 1991b] Scholtes, J.C., "Learning Simple Semantics by Self-Organization", *AAAI Spring Symposium Series*, Stanford University, pp. 78-83, 26-28 March 1991.

[Scholtes, 1991c] Scholtes, J.C., "Recurrent Kohonen Self-Organization in Natural Language Processing", *Proceedings of the ICANN'91*, Helsinki, Finland, 24-28 June 1991.

[Scholtes, 1991d] Scholtes, J.C., "Kohonen Feature Maps in Natural Language Processing", ITLI Prepublication Series for Computational Linguistics CL-91-01, University of Amsterdam, October 1991.

[Schulenburg, 1992] Schulenburg, D., "Sentence Processing with Realistic Feedback", *Proceedings of the IJCNN'92*, Baltimore, MD, USA, Vol. IV, pp. 661-666, June 1992.

[Servan-Schreiber *et al*, 1988] Servan-Schreiber, D., Cleeremans, A., & McClelland, J.L., "Encoding Sequential Structure in Simple Recurrent Networks", Technical Report, CMU-CS-88-183, Carnegie Mellon University, 1988.

[Servan-Schreiber *et al*, 1989] Servan-Schreiber, D., Cleeremans, A., & McClelland, J.L., "Learning Sequential Structure in Simple Recurrent Networks", *Advances in Neural Information Processing Systems 1*, pp. 643-652, 1989.

[Servan-Schreiber *et al*, 1991] Servan-Schreiber, D., Cleeremans, A., & McClelland, J.L., "Graded State Machines: The Representation of Temporal Contingencies in Simple Recurrent Networks", *Machine Learning*, Vol. 7, pp. 161-193, 1991.

[Silva & Almeida, 1990] Silva, F.M., & Almeida, L.B., "Speeding up backpropagation", in *Advanced Neural Computers*, ed. Eckmiller, R., Elsevier Science Publishers, Amsterdam, 1990.

[Sompolinsky & Kanter, 1986] Sompolinsky, H., & Kanter, I., "Temporal Association in Asymmetric Neural Networks", *Physical Review Letters*, Vol. 57, pp. 2861-2864, 1986.

[Stornetta *et al*, 1988] Stornetta, W.S., Hogg, T., & Huberman, B.A., "A Dynamical Approach to Temporal Pattern Processing" *Advances in Neural Information Processing Systems*, ed. D. Anderson, New York: American Institute of Physics, pp. 750-759, 1988.

[Sutton, 1987] Sutton, R., "Learning to Predict by the Methods of Temporal Differences", GTE Laboratories Technical Report, TR87-5091, 1987.

[Szu, 1986] Szu, H., "Fast Simulated Annealing", *AIP Conference Proceedings 151: Neural Networks for Computing*, ed. J. Denker, New York: American Institute of Physics, pp. 420-425, 1986.

[Toomarian & Barhen, 1991] Toomarian, N., & Barhen, J., "Fast Temporal Neural Learning Using Teacher Forcing" *Proceedings of the IJCNN'91*, Seattle, pp. I: 817-822, 1991.

[Unnikrishnan *et al*, 1988] Unnikrishnan, K.P., Hopfield, J.J., & Tank, D.W., "Learning Time-delayed Connections in a Speech Recognition Circuit", *Proceedings of the Neural Networks for Computing Conference*, Snowbird, Utah, 1988.

[Vapnik, 1982] Vapnik, V., *Estimation of Dependences Based on Empirical Data*, Springer-Verlag, 1982.

[Von der Malsburg, 1973] Von der Malsburg, C., "Self-Organization of Orientation Sensitive Cells in the Striate Cortex", *Kybernetik*, Vol. 14, pp. 85-100, 1973.

[Waibel *et al*, 1988] Waibel, A., Sawai, H., & Shikano, K., "Modularity and Scaling in Large Phonemic Neural Nets", Technical Report TR-I-0034, ATR Interpreting Telephony Research Laboratories, Japan, 1988.

[Wang *et al*, 1993] Wang, M., Liu, W., & Zhong, Y., "Simple Recurrent Network for Chinese Word Prediction" *Proceedings of the IJCNN'93*, Nagoya, Japan, Vol. I, pp. 263-266, October 1993.

[Watrous & Shastri, 1987] Watrous, R.L., & Shastri, L., "Learning Phonetic Features using Connectionist Networks: An Experiment in Speech Recognition", in *Proceedings of the IEEE First ICNN*, San Diego, CA, Vol. II, pp. 619-627, June 1987.

[Watrous, 1988] Watrous, R.L., *Speech Recognition Using Connectionist Networks*, Ph.D thesis, University of Pensylvania, 1988.

[Widrow, 1962] Widrow, B., "Generalization and Information Storage in Networks of Adline 'Neurons"', *Self-Organizing Systems - 1962*, eds. M. Yovits, G. Jacobi and G. Goldstein, Washington:Spartan Books, pp. 435-461, 1962.

[Williams & Zipser, 1989] Williams, R.J., and Zipser, D., "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks" *Neural Computation*, Vol. 1, pp. 270-280, 1989.

[Williams & Zipser, 1990a] Williams, R.J., and Zipser, D., "Gradient-Based Learning Algorithms for Recurrent Connectionist Networks", Technical Report NU-CCS-90-9, College of Computer Science, Northeastern University, Boston, April 1990.

[Williams & Peng, 1990b] Williams, R.J., & Peng, J., "An Efficient Gradient-Based Algorithm for On-line Training of Recurrent Network Trajectories", *Neural Computation*, Vol. 2, pp. 490-501, 1990.

[Winter, 1991] Winter, C.L., "An Adaptive Network that Learns Sequences of Transitions", *Intelligent Engineering Systems through Artificial Neural Networks*, eds. Dagli, C.H., Kumara, S.R.T., Shin, Y.C., ASME Press, ANNIE-91, St. Louis, Missourri, pp. 653-660, November 1991.

[Zipser, 1989] Zipser, D., "A Subgrouping Strategy that Reduces Complexity and Speeds UP Learning in Recurrent Networks", *Neural Compuiation*, Vol. 1, pp. 552-558, 1989.

[Zurada, 1992] Zurada, J.M., In *Introduction to Artificial Neural Systems*, Chapter 4: "Multilayer Feedforward Networks", West Publishing Company, St. Paul, pp. 163-248, 1992.

Appendix A

Transfer Functions and Recurrent Algorithms

In this appendix we outline different transfer functions and recurrent supervised algorithms that can be used in architecture-specific recurrent neural networks.

A.1 Transfer functions

Transfer functions compute the activation of a unit, given its inputs. It can be decomposed into a similarity measure and an activation function.

The *similarity measure* $d_i(t)$ measures the similarity between the input activity pattern $x(t) = \{x_1(t), \ldots, x_j(t), \ldots, x_N(t)\}$ and the weights $w_i(t) = \{w_{1i}, \ldots, w_{ji}, \ldots, w_{Ni}\}$ connecting the unit i to N units of the input activities. The threshold $\theta_i(t)$ is associated with unit i and determines the scale of the similarity measure $d_i(t)$. Two common similarity measures are Inner product (1) and Euclidean distance (2).

$$d_i(t+1) = \sum_{j=1}^{N} w_{ji}(t)x_j(t) - \theta_i(t) \qquad (1)$$

$$d_i(t+1) = (\sum_{j=1}^{N}(w_{ji}(t) - x_j(t))^2)^{1/2} - \theta_i(t) \qquad (2)$$

The *activation function* $F(d_i(t))$ maps the similarity measure $d_i(t)$ to a prespecified activation range $x_i(t)$. Five common examples are the Linear (3), Ramp Threshold (4), Step Threshold (5), Gaussian (6) and Sigmoid (7) activation functions (β denotes a constant).

$$x_i(t) = F(d_i(t)) = d_i(t) \qquad (3)$$

$$x_i(t) = F(d_i(t)) = \begin{cases} 0 & \text{for } d_i(t) < 0 \\ d_i(t) & \text{for } d_i(t) \geq 0 \end{cases} \qquad (4)$$

$$x_i(t) = F(d_i(t)) = \begin{cases} 0 & \text{for } d_i(t) < 0 \\ 1 & \text{for } d_i(t) \geq 0 \end{cases} \qquad (5)$$

$$x_i(t) = F(d_i(t)) = exp(-\beta d_i^2(t)) \qquad (6)$$

$$x_i(t) = F(d_i(t)) = \frac{1}{1 + exp(-\beta d_i(t))} \qquad (7)$$

A.2 Recurrent supervised learning algorithms

In a recurrent supervised learning task the weights are changed according to some performance measure of the error $e_i(t)$ between the computed output unit values $x_i(t)$ and the desired (target) values $s_i(t)$ at specified times t (see equation (8)).

$$e_i(t) = \begin{cases} s_i(t) - x_i(t) & \text{for } i \text{ an output unit} \\ 0 & \text{otherwise} \end{cases} \qquad (8)$$

The weight change equations of the following recurrent supervised learning algorithms will be given: real-time backpropagation-through-time (BPTT), epochwise BPTT, associative BPTT (Recurrent backpropagation), truncated BPTT, epoch-truncated BPTT, real-time recurrent learning (RTRL), and forward propagation BPTT (FP-BPTT). We conclude this section with a comparison of these algorithms.

Real-Time BPTT:

In real-time BPTT the weights are changed at each time step while the network continues to run. The values $\delta_i(\tau)$ (the error signal term (*delta*) used for weight updating), which are needed in (10) to calculate the weight changes Δw_{ji}, are computed for each fixed t, all $i \in U$ (where U denotes the set of all the units, except the inputs) and $\tau \in (t_0, t]$ using the following equation

$$\delta_i(\tau) = \begin{cases} F_i'(d_i(\tau))e_i(\tau) & \text{if } \tau = t \\ \sum_{j \in U} w_{ji}\delta_j(\tau + 1) & \text{if } t_0 < \tau < t \end{cases} \qquad (9)$$

$$\Delta w_{ji} = \alpha \sum_{\tau=t_0+1}^{t} \delta_j(\tau)x_i(\tau - 1) \qquad (10)$$

Epochwise BPTT:

In epochwise BPTT the training stream is segmented into epochs, where t_0 denotes the start time of the epoch and t_1 denotes the end time. The weights are changed (according to equation (12)), after computing the $\delta_i(\tau)$ values for each fixed t, all $i \in U$ and $\tau \in (t_0, t_1]$ by means of the following equation:

$$\delta_i(\tau) = \begin{cases} F_i'(d_i(\tau))e_i(\tau) & \text{if } \tau = t_1 \\ F_i'(d_i(\tau))e_i(\tau) + \sum_{j \in U} w_{ji}\delta_j(\tau + 1) & \text{if } t_0 < \tau < t_1 \end{cases} \qquad (11)$$

$$\Delta w_{ji} = \alpha \sum_{\tau=t_0+1}^{t_1} \delta_j(\tau)x_i(\tau - 1) \qquad (12)$$

Associative BPTT (Recurrent Backpropagation):

The associative BPTT, also called recurrent backpropagation (RBP), is a special case of epochwise BPTT for situations when both the computed and desired trajectories consist of settling to a stable state. The network is driven with a constant input, and the network state at the end of each epoch $[t_0, t_1]$ is compared with some desired state. The $\delta_i(\tau)$ values are obtained by evaluating the equation (13) for $\tau \in (t_0, t_1]$. Weight changes are made via (14).

$$\delta_i(\tau) = \begin{cases} F_i'(d_i(\tau))e_i(\tau) & \text{if } \tau = t_1 \\ \sum_{j \in U} w_{ji}\delta_j(\tau + 1) & \text{if } t_0 < \tau < t_1 \end{cases} \qquad (13)$$

$$\Delta w_{ji} = \alpha x_i(t_1) \sum_{\tau=t_0+1}^{t_1} \delta_j(\tau) \qquad (14)$$

Truncated BPTT:

The truncated BPTT, denoted by BPTT(h) is a bounded-history approximation of the real-time BPTT in which relevant network information is saved for a fixed number h of time steps and any information older forgotten. The $\delta_i(\tau)$ values are computed in a similar manner as in real-time BPTT, but only for $\tau \in (t-h, t]$ (15). The weight changes are calculated by means of equation (16).

$$\delta_i(\tau) = \begin{cases} F_i'(d_i(\tau))e_i(\tau) & \text{if } \tau = t \\ F_i'(d_i(\tau)) \sum_{j \in U} w_{ji}\delta_j(\tau+1) & \text{if } t-h < \tau < t \end{cases} \quad (15)$$

$$\Delta w_{ji} = \alpha \sum_{\tau=t-h+1}^{t} \delta_j(\tau) x_i(\tau-1) \quad (16)$$

Epoch-truncated BPTT:

The epoch-truncated BPTT, denoted by BPTT($h; h_1$), is obtained by combining aspects of epochwise and truncated BPTT. The network is run through h_1 additional time steps before the next BPTT computation, where $h_1 \leq h$. It approximates the error gradient in the weight space by taking into account only the $[t-h_1, t]$ part of the history over the interval $[t-h, t]$. The $\delta_i(\tau)$ values are computed in a similar manner as truncated BPTT, but only for $\tau \in (t-h, t]$ using (17). After this backward pass has been completed, the weight changes are calculated by means of equation (18).

$$\delta_i(\tau) = \begin{cases} F_i'(d_i(\tau))e_i(\tau) & \text{if } \tau = t \\ F_i'(d_i(\tau))e_i(\tau) + \sum_{j \in U} w_{ji}\delta_j(\tau+1) & \text{if } t-h_1 < \tau < t \\ F_i'(d_i(\tau)) \sum_{j \in U} w_{ji}\delta_j(\tau+1) & \text{if } t-h < \tau < t-h_1 \end{cases} \quad (17)$$

$$\Delta w_{ji} = \alpha \sum_{\tau=t-h+1}^{t} \delta_j(\tau) x_i(\tau-1) \quad (18)$$

RTRL:

The RTRL algorithm computes the error gradient by propagating activity gradient information forward, as opposed to BPTT, which uses the backward propagation of error information. In order to obtain the activity gradient information, the $p_{ji}^k(t)$ values are defined for each $k \in U$, $j \in U \bigcup I$ (where I is the set of input units), $i \in U$, and $t_0 \leq t \leq t_1$ in the following manner,

$$p_{ji}^k(t) = \frac{\partial x_k(t)}{\partial w_{ji}} \quad (19)$$

The $p_{ji}^k(t)$ values are computed at each time step by means of equation (20). The weights are changed by combining the $p_{ji}^k(t)$ values with the error values $e_k(t)$ via equation (21).

$$p_{ji}^k(t+1) = F_k'(d_k(t+1))[\sum_{l \in U} w_{kl} p_{ji}^l + \delta_{jk}(t) x_i(t)] \qquad (20)$$

$$\Delta w_{ji} = \alpha \sum_{k \in U} e_k(t) p_{ji}^k(t) \qquad (21)$$

FP-BPTT:

The FP-BPTT algorithm, denoted by FP-BPTT(h), combines aspects of both the forward propagation used in RTRL and the backward error propagation used in the BPTT algorithm. Time is decomposed into disjoint intervals each of length $h = t - t'$. The $\delta_k(\tau)$ values are computed in a similar manner as in epochwise BPTT over the interval $[t',t]$ using equation (22).

$$\delta_k(\tau) = \begin{cases} F_k'(d_k(\tau)) e_k(\tau) & \text{if } \tau = t \\ F_k'(d_k(\tau)) e_k(\tau) + \sum_{l \in U} w_{lk} \delta_l(\tau + 1) & \text{if } \tau < t \end{cases} \qquad (22)$$

The weight changes are performed only at the end of each interval, using all the $p_{ji}^k(t)$ values (as used in RTRL) and the $\delta_k(\tau)$ values, by means of the following equation

$$\Delta w_{ji} = \alpha \sum_{l \in U} e_l(t') p_{ji}^l(t') + \sum_{\tau=t'}^{t-1} \delta_j(\tau + 1) x_i(\tau) \qquad (23)$$

In order to compute the $p_{ji}^r(t)$ values ($r \in U$) for the next time interval, the algorithm first performs a BPTT computation using equation (22) to obtain $\delta_i(\tau)$ values for $t' < \tau \leq t$ together with the following equation

$$\varepsilon_k(\tau) = \begin{cases} \delta_{kr} & \text{if } \tau = t \\ \sum_{l \in U} w_{lk} \delta_l(\tau + 1) & \text{if } \tau < t \end{cases} \qquad (24)$$

to obtain $\varepsilon_k(\tau)$ for $t' \leq \tau \leq t$. Note that δ_{kr} is the Kronecker delta and must not be confused with δ_l. The $p_{ji}^r(t)$ values for each i and j can be computed via the following equation

$$p_{ji}^r(t) = \alpha \sum_{l \in U} e_l(t') p_{ji}^l(t') + \sum_{\tau=t'}^{t-1} \delta_j(\tau + 1) x_i(\tau) \qquad (25)$$

Comparison of Recurrent Algorithms:

Table A.1 gives a comparison of the worst-case (fully connected network with all weights adaptable) storage complexity (in terms of the number of real numbers stored), time complexity (in terms of the number arithmetic operations required), exact or approximate gradient, and online versus offline computation (see page 9). For this comparison n denotes the number of noninput units, k the number of time steps, L the total time over which the network is run, k_1 the number of additional time steps (where $k_1 \leq k$), g the number of subnetworks, and c a constant.

LEARNING ALGORITHM	STORAGE COMPLEXITY	TIME COMPLEXITY	GRADIENT	ONLINE
Realtime BPTT	$O(nL)$	$O(n^2L)$	exact	yes
Epochwise BPTT	$O(nk)$	$O(n^2)$	exact	no
Truncated BPTT(k)	$O(nk)$	$O(n^2k)$	approx.	yes
Epoch-truncated BPTT($k;k_1$)	$O(nk)$	$O(n^2k/k_1)$	approx.	yes
BPTT($k;ck$)	$O(nk)$	$O(n^2)$	approx.	yes
RTRL	$O(n^3)$	$O(n^4)$	exact	yes
Subgrouping RTRL(g)	$O(n^3/g)$	$O(n^2/g^2)$	approx.	_ s
RTRL(cn)	$O(n^2)$	$O(n^2)$	approx.	yes
Hybrid FP-BPTT(k)	$O(n^3+nk)$	$O(n^3+n^4/k)$	exact	both
FP-BPTT(cn)	$O(n^3)$	$O(n^3)$	exact	both

Table A.1: Worst-case storage and time complexities, exact or approximate gradient, and online versus offline computation for recurrent algorithms

Appendix B

Training Strategies

The simulation results of the initial experiments for Increased Complexity Training (see section 3.2.1) are summarised in Tables B.1 and B.2. For Table B.1 the maximum success ratio obtained on the training set is given in brackets. The results of the *incremental success ratio* (*isr*) training are given in Table B.2. In this case ICT improves training time by 50% (35 epochs vs. 71) compared to training on a fixed set, and by 34% (35 epochs vs. 53) over its previous value when *isr* is used. For CST comparison with its previous result (not using *isr*) shows a 26% improvement, but again the total number of epochs required is much more than that using ICT. It seems as if a doubling in complexity using an incremental success ratio and increased complexity training only produces a linear increase in the number of epochs.

# of patterns		Fixed	ICT		CST
10	set0-1		12 (100%)	random1	95 (75%)
20	set0-3		8 (100%)	random12	115 (98%)
40	set0-7		14 (100%)	random123	3 (98%)
80	set0-15	71 (100%)	19 (100%)	random1234	69 (100%)
Total epochs		71	53		282
Success ratio	test set	100%	100%		100%

Table B.1: Number of epochs for the different strategies

Table B.3 presents the training strategy results for the Digital Morse Code Generation experiment using the Elman ASRNN, whereas Table B.4 presents the results for the feedforward Digit Recognition experiment.

# of patterns		Fixed	ICT (*isr*)		CST (*isr*)
10	set0-1		2 (70%)	random1	81 (70%)
20	set0-3		6 (93%)	random12	35 (93%)
40	set0-7		12 (90%)	random123	24 (90%)
80	set0-15	71 (100%)	15 (100%)	random1234	68 (100%)
Total epochs		71	35		208
Success ratio	test set	100%	100%		100%

Table B.2: Number of epochs using *isr*

Training Strategies	Increment	Updates	Improved
ISDST - Incremental Smallest Delta Subset Training	2	12690	76.3%
ILDST – Incremental Largest Delta Subset Training	3	15465	71.1%
IST – Incremental Subset Training	1	16950	68.3%
SDST – Smallest Delta Subset Training	-	30900	42.2%
LDST – Largest Delta Subset Training	-	35000	34.6%
IADST – Incremental Alternating Delta Subset Training	5	38675	27.7%
CST – Combined Subset Training	-	52650	2%
FST – Fixed Set Training	-	53500	-

Table B.3: Performance of training strategies for Digital Morse Code Generation (Elman AS-RNN) – see section 3.4.1

Training Strategies	Increment	Updates	Improved
ISDST - Incremental Smallest Delta Subset Training	2	14524	58.4%
ISDSET - Incremental Smallest Delta Subset Epoch Training	5	16612	52.4%
ISET - Incremental Subset Epoch Training	5	19240	45.0%
IST - Incremental Subset Training	3	21789	38.0%
ILDSET - Incremental Large Delta Subset Epoch Training	4	26829	23.1%
CSET - Combined Subset Epoch Training	-	26999	22.6%
IADSET - Incremental Alternating Delta Subset Epoch Training	12	27108	22.3%
IADST - Incremental Alternating Delta Subset Training	11	28081	19.5%
SDSET - Smallest Delta Subset Epoch Training	-	31266	10.4%
EFST - Epoch Fixed Set Training	-	32806	6.0%
ADST - Alternating Delta Subset Training	-	32983	5.5%
SDST - Smallest Delta Subset Training	-	33065	5.3%
ILDST - Incremental Large Delta Subset Training	1	33412	4.3%
FST - Fixed Set Training	-	34900	-

Table B.4: Performance of the Training Strategies for Digit Recognition (Feedforward BP) – see section 3.4.3

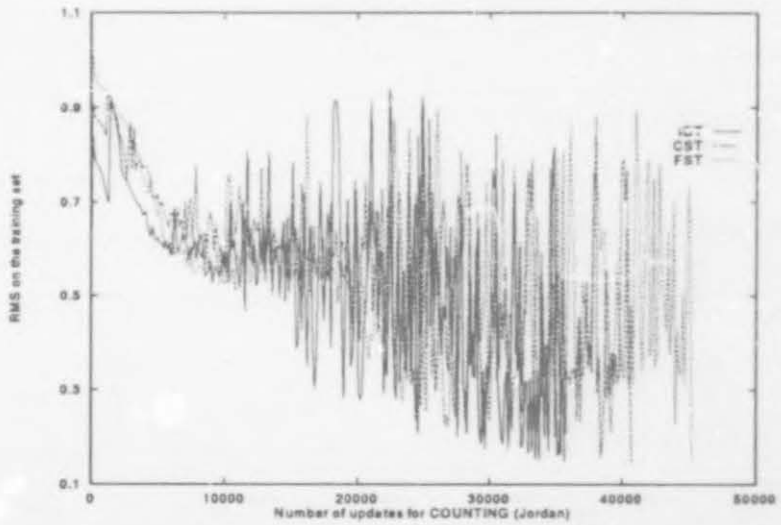


Figure B.1: Performance of the Jordan ASRNN for Counting – see section 3.2.3

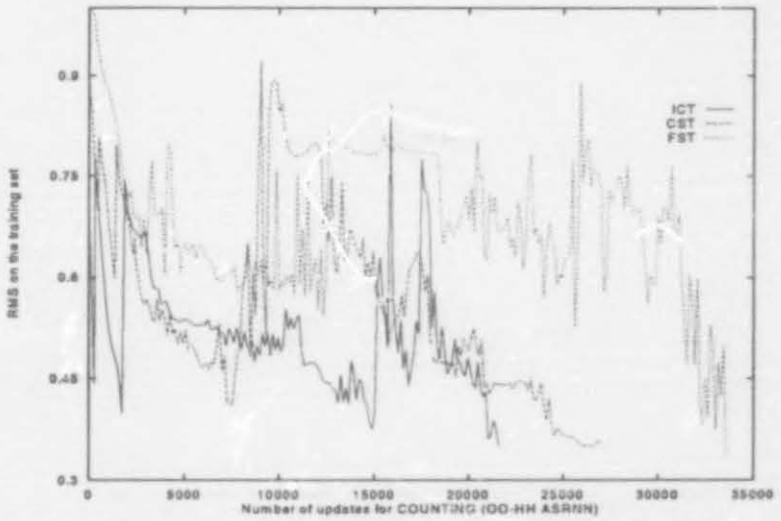


Figure B.2: Performance of the Output-to-Output Hidden-to-Hidden ASRNN for Counting – see section 3.2.5

```
while not done do begin
begin
  output(WRITE, low_order_result);
  if sum>radix then output(CARRY,'00');
  output(NEXT,'00');
end
if carry_on_previous_input then output(WRITE,'01');
output(DONE,'00');
```

$$\begin{array}{rr} & 203_4 \\ + & 313_4 \\ \hline & 1122_4 \end{array}$$

Input			Output					
			Action				Result	
EOI	I_1	I_2	W	C	N	D	O_1O_2	
0	11	11	1	0	0	0	10	Write result $3_4+3_4=2_4+C_{Out}$
0	11	11	0	1	0	0	00	Record carry
0	11	11	0	0	1	0	00	Next column
0	00	01	1	0	0	0	10	Write result $0_4+1_4+C_{In}=2_4$
0	00	01	0	0	1	0	00	No carry, next column
0	10	11	1	0	0	0	01	Write result $2_4+3_4=1_4+C_{Out}$
0	10	11	0	1	0	0	00	Carry
0	10	11	0	0	1	0	00	Next
0	10	11	1	0	0	0	01	Write
1	10	11	0	0	0	1	00	Done

Figure B.3: Addition program code [Cottrell & Tsung, 1991] and an example of temporal sequences involved in 3-column Addition – see section 3.2.8

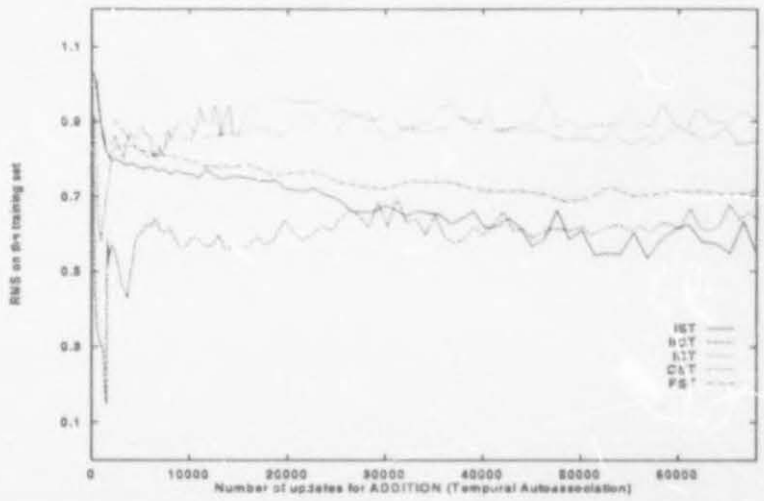


Figure B.4: Performance of the Temporal Autoassociation ASRNN for Addition – see section 3.2.9

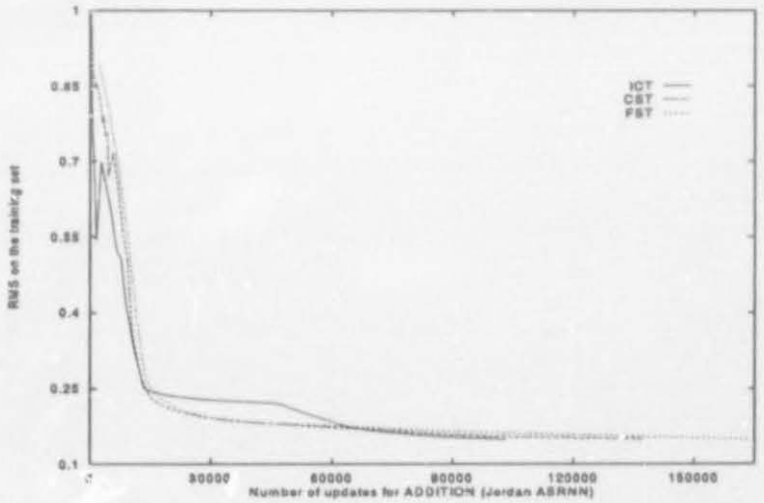


Figure B.5: Performance of the Jordan ASRNN for Addition – see section 3.2.10

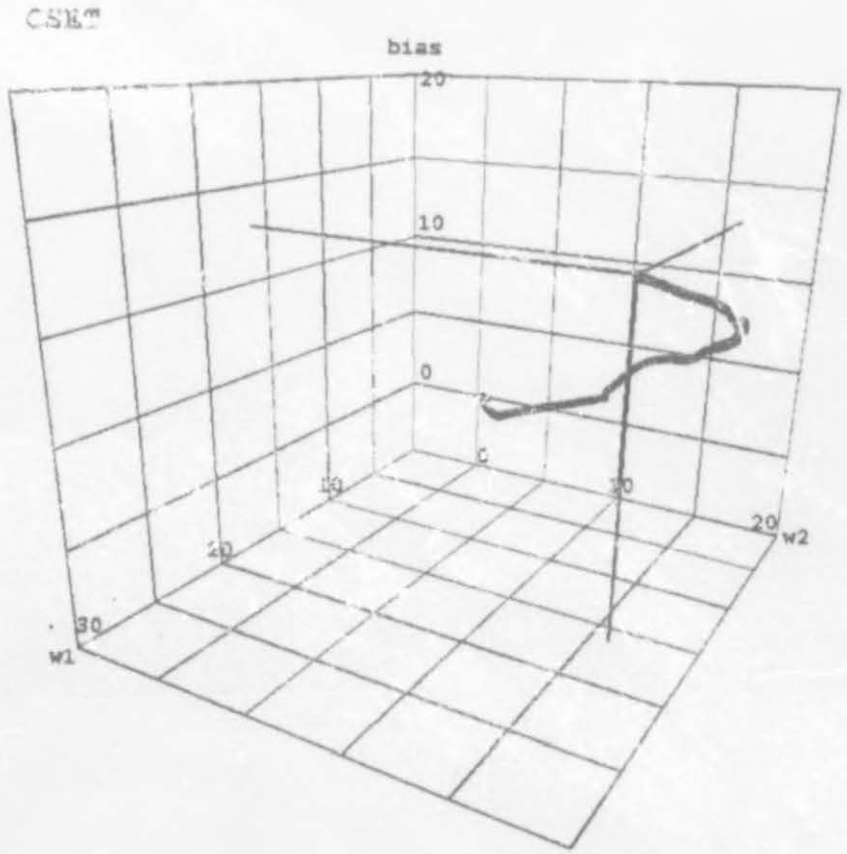


Figure B.6: Visualization of the movement of the weights in the weight space during CSET for Cluster Detection – see section 3.2.11

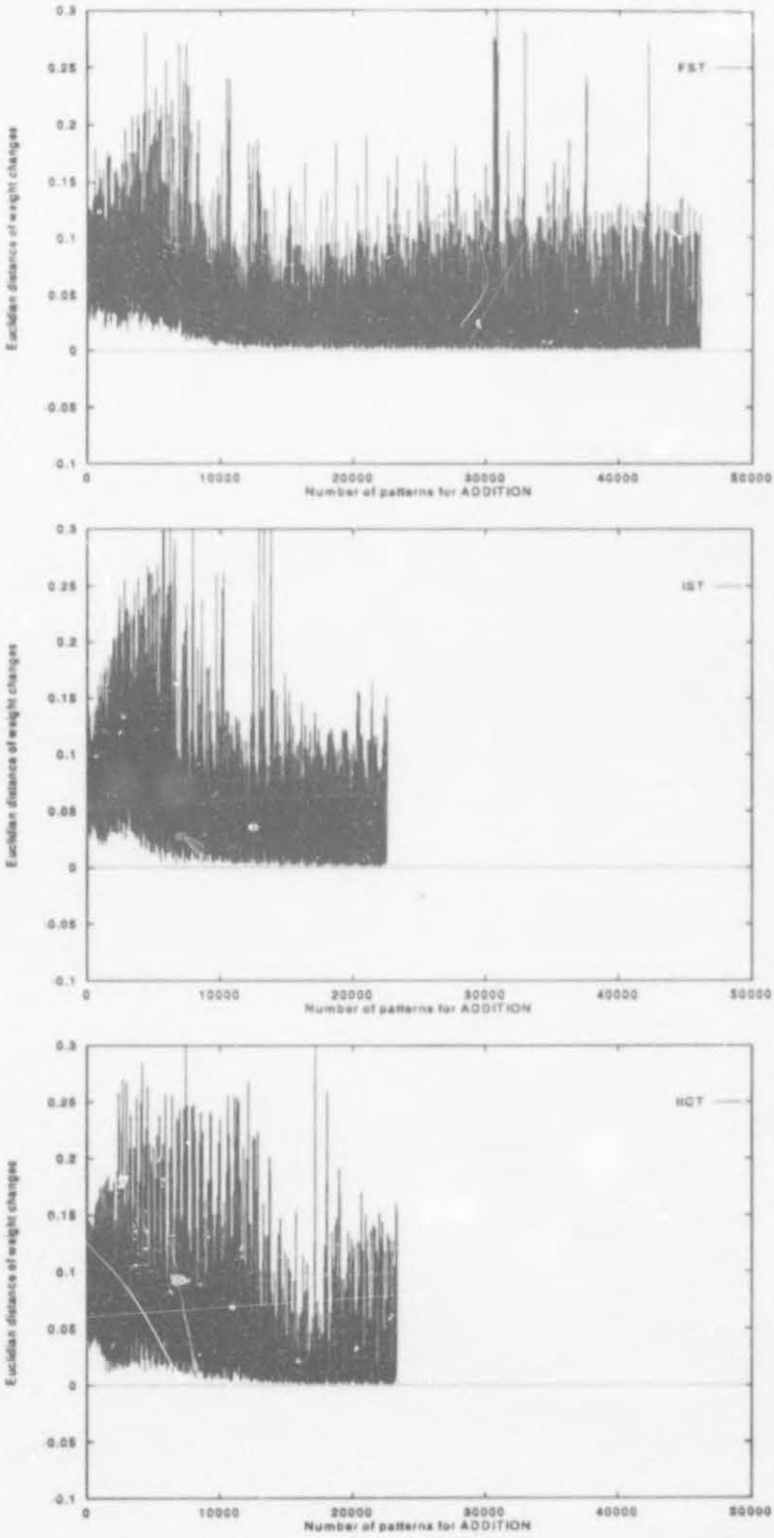


Figure B.7: Visualization of the Euclidean distance of the weight changes of a particular weight during FST, IST and IICT training for Addition (Elman ASRNN) – see section 3.3.1

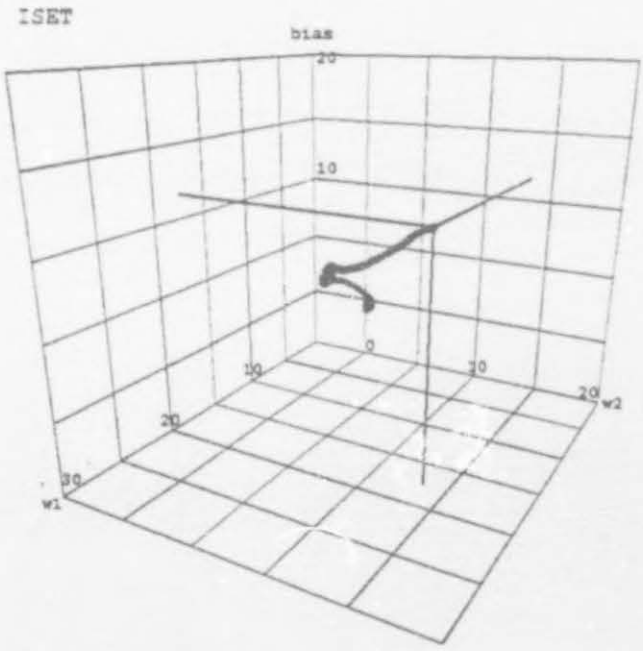


Figure B.8: Visualization of the movement of the weights in the weight space during ISET for Cluster Detection – see section 3.3.4

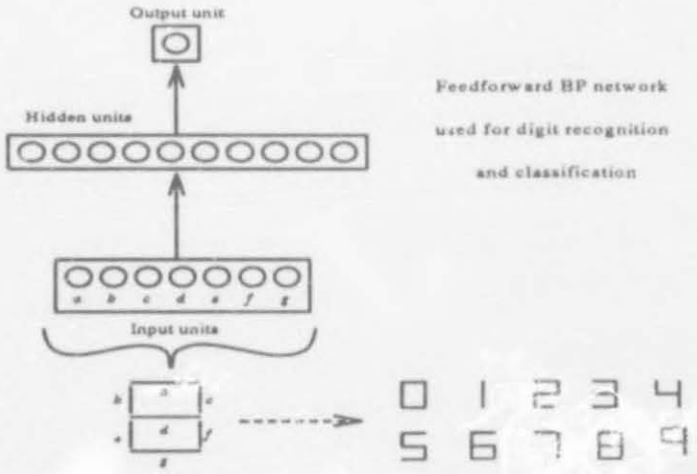


Figure B.9: The Elman ASRNN used for Digit Recognition – see section 3.4.3

Appendix C

Dynamics Analysis of Classification and Learning

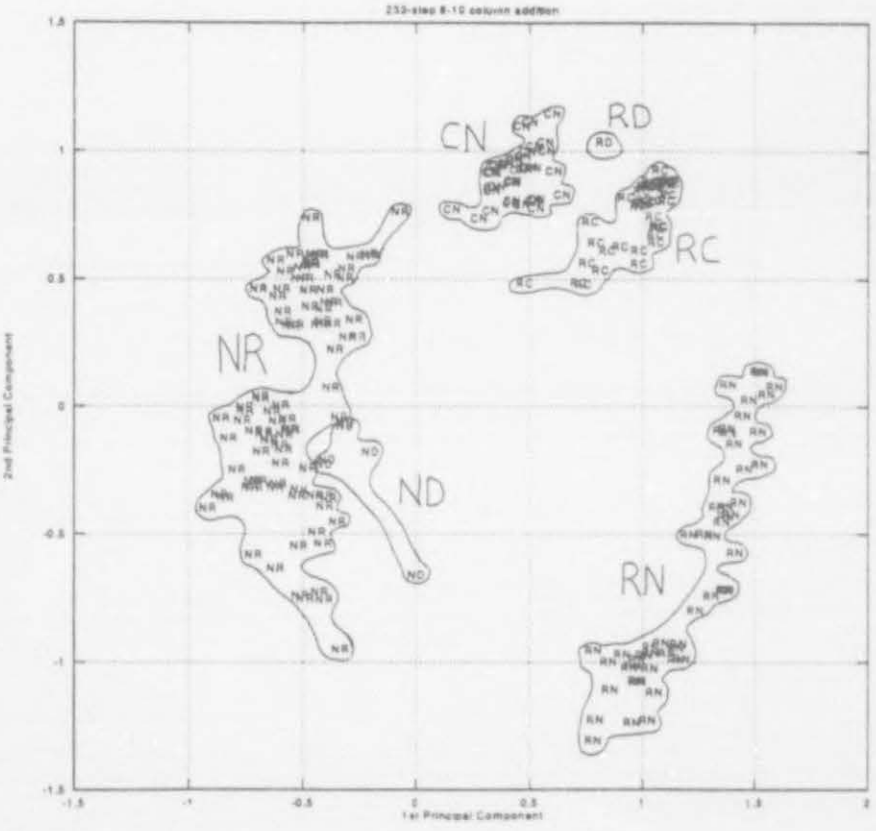


Figure C.1: Principal Component Analysis of 233-step 8-10 column addition – see section 4.2.6.4

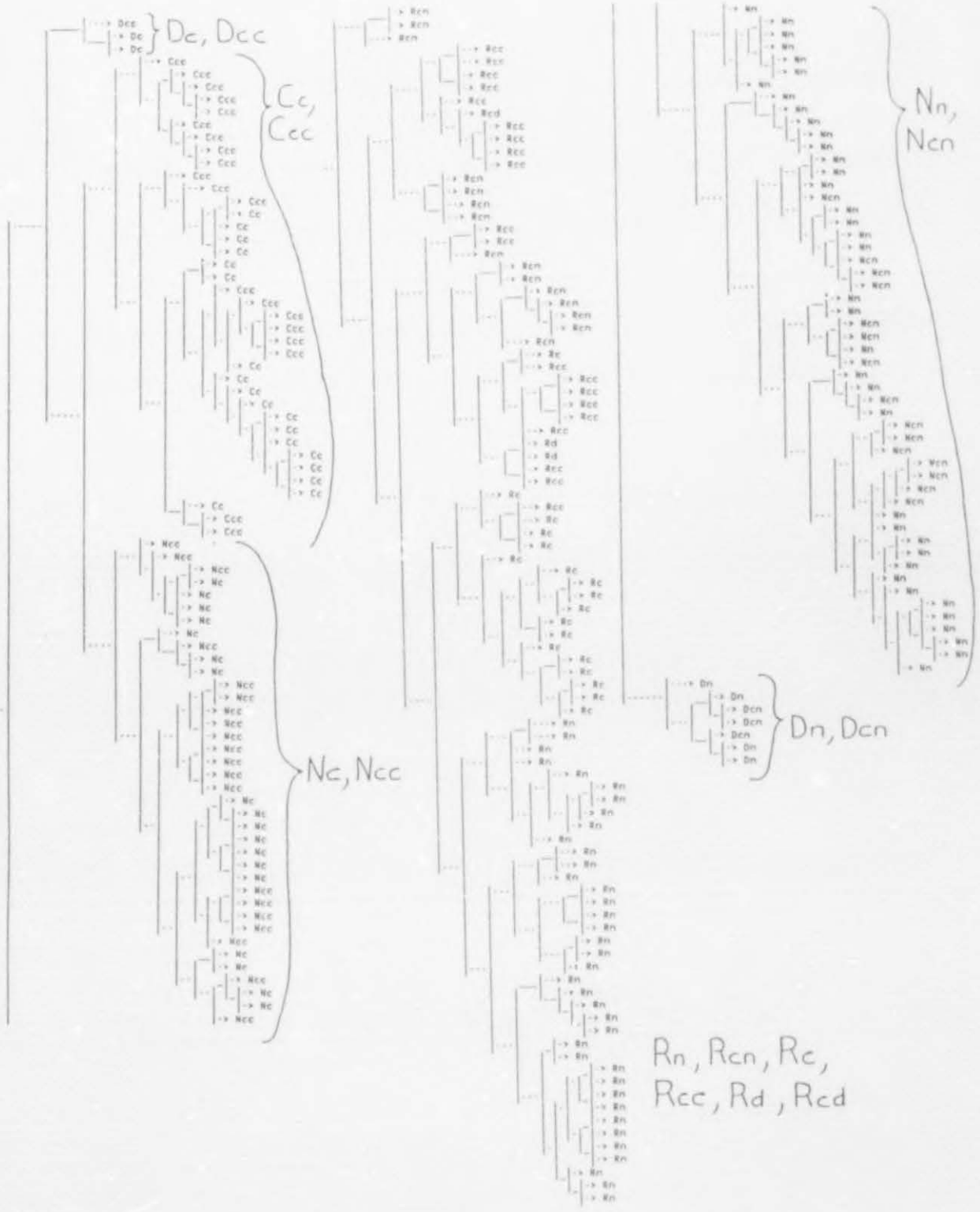


Figure C.2: Hierarchical Cluster Analysis of 233-step 8-10 column addition (Mealy machine correspondence) – see section 4.2.6.4

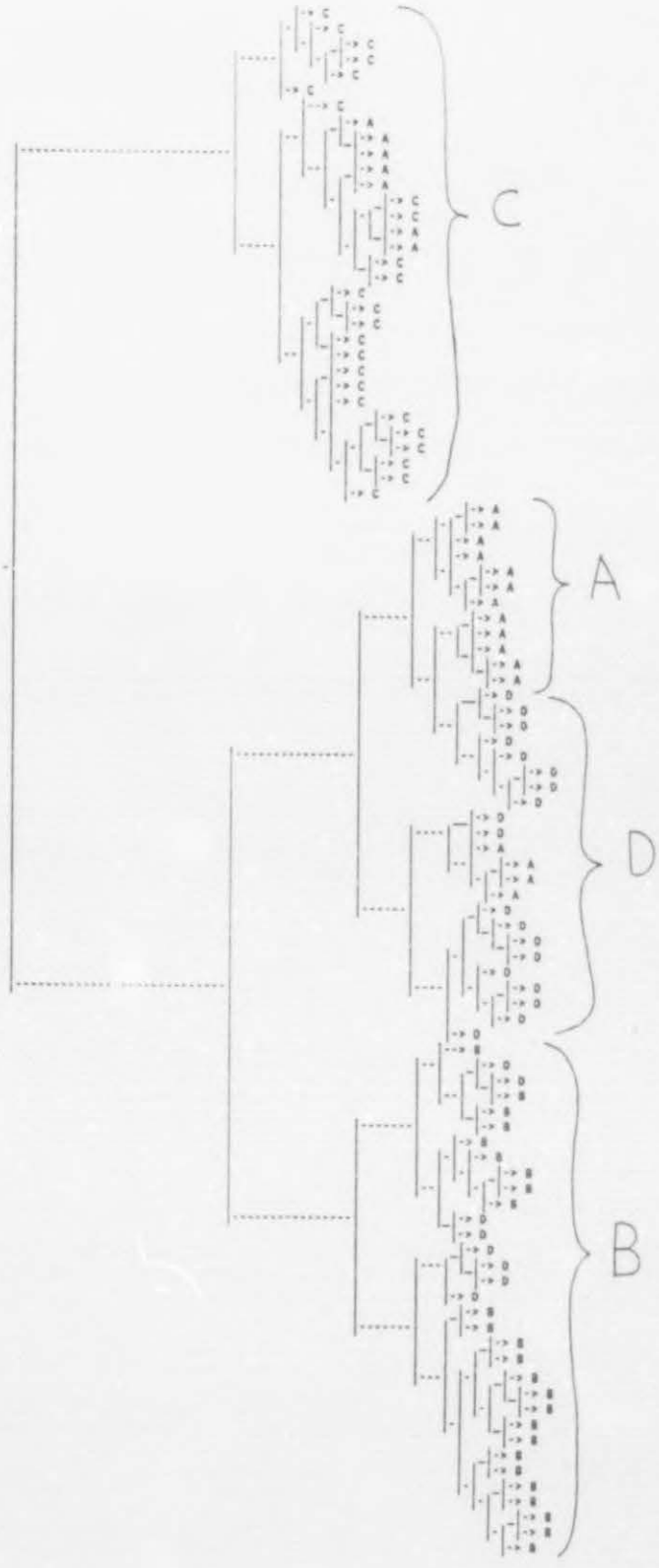


Figure C.3: Hierarchical Cluster Analysis of Temporal $\overline{XOR}$ classification – see section 4.2.6.1

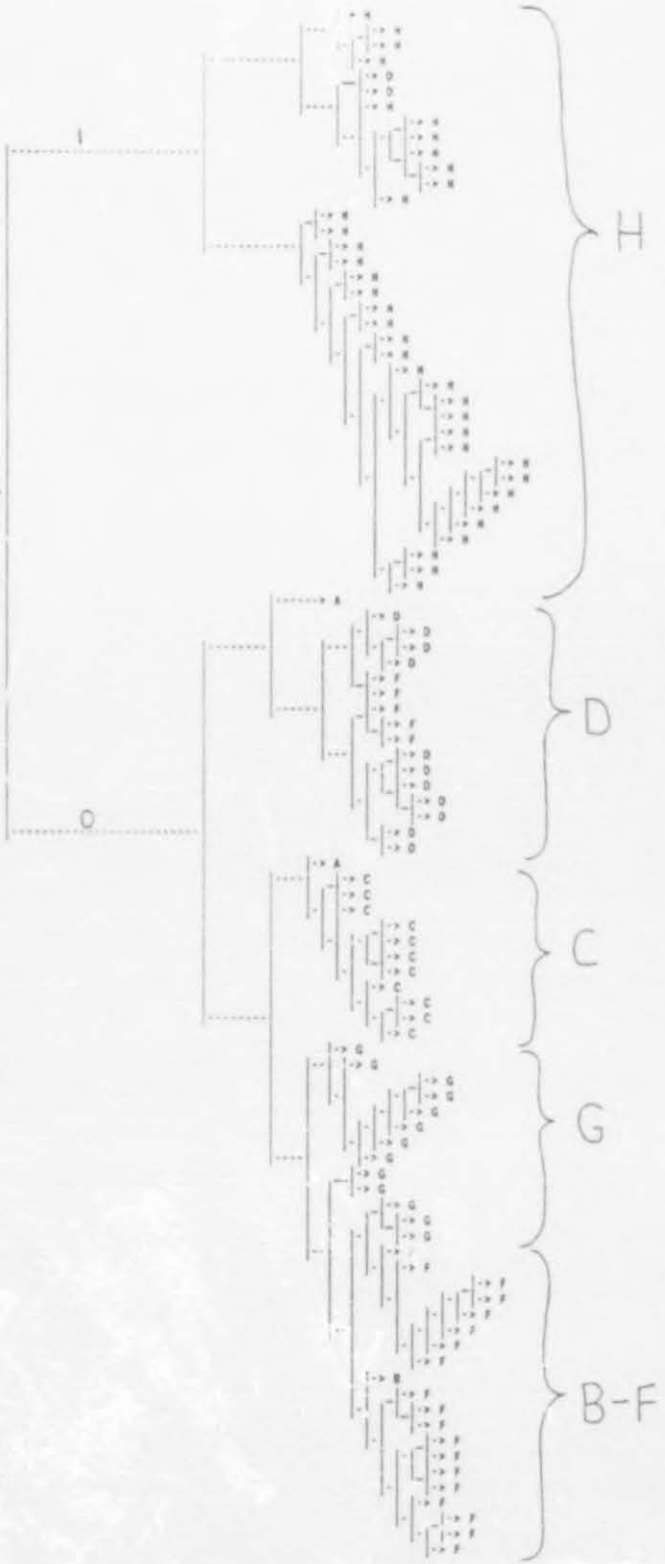


Figure C.4: Hierarchical Cluster Analysis of the detection of three consecutive ones – see section 4.2.6.3

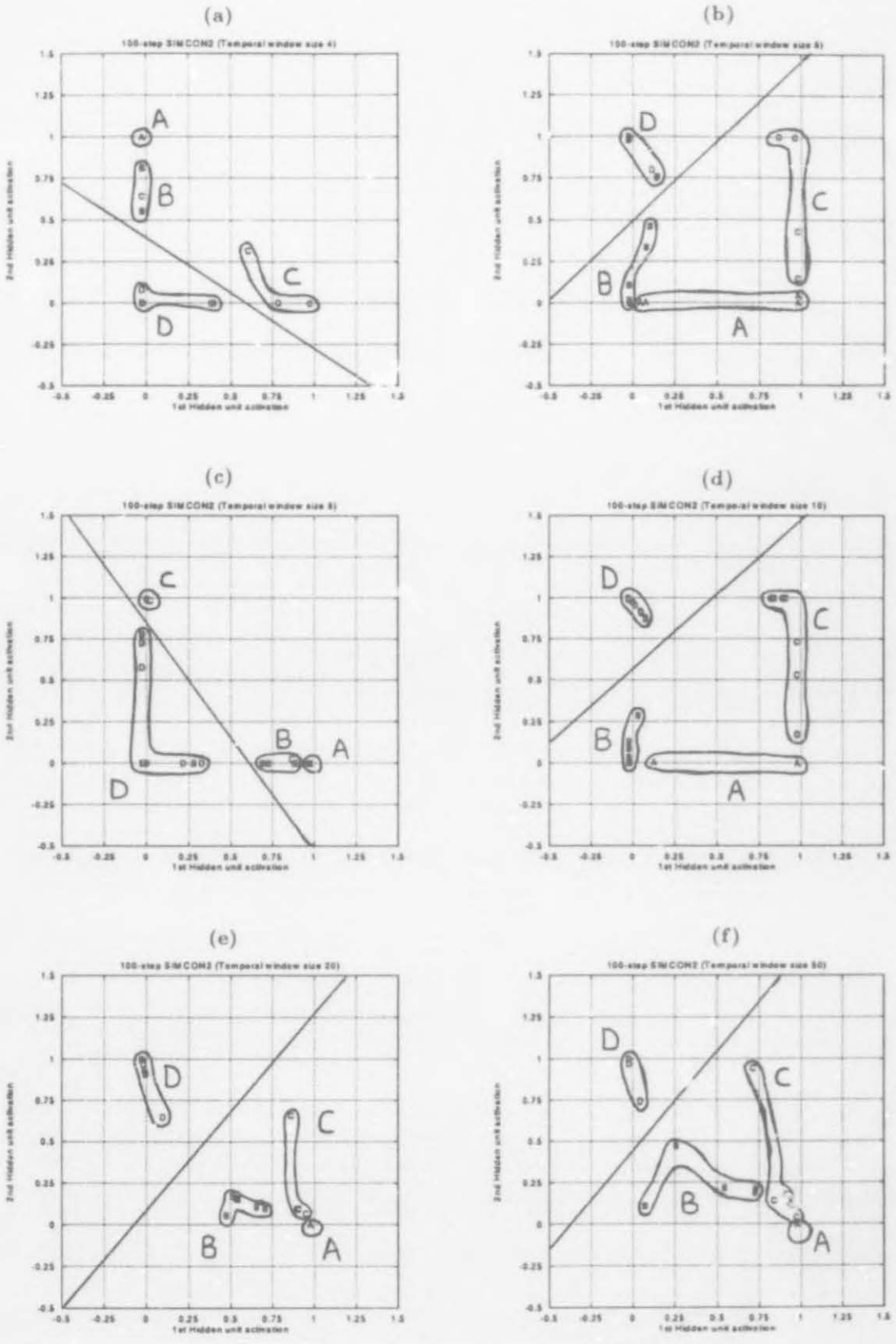


Figure C.5: Visualization of hidden activation space for varying temporal window size 4 to 50 – see section 4.2.7.4

Appendix D

Bounds for the Capacity and Number of Hidden Units

D.1 Capacity

D.1.1 n:h:1 Upper bound for Elman ASRTNs

Mitchison & Durbin (1989) proved that $O(nh\log_2 h)$ is an upper bound for the probabilistic capacity of *n:h:1* FFTNs. In a similar fashion an upper bound which is an order estimation of the probabilistic capacity of *n:h:1* Elman ASRTNs is next proved.

Theorem D.1:

The probabilistic capacity of an Elman ASRTN with n inputs, a single hidden layer of h threshold units, a single context layer of h previous hidden layer values, and one threshold output unit is at most $O((n+h+1)h\log_2 h)$ examples in general position, where $n+h \leq N$, $h \leq n$ and $N \geq (2+\sqrt{2})(n+h+1)+2$.

Proof:

The objective in this proof is to obtain that N for which $D(N, n+h+1)^h = 2^{(N-1)}$ gives an upper bound capacity for an *n:h:1* Elman ASRTN. We are going to replace $D(N, n+h+1)$ by larger expressions and eventually obtain an upper bound for which $D(N, n+h+1) \geq 2^{N-1}$. This will also be an upper bound for a solution to $D(N, n+h+1)^h = 2^{N-1}$, since for small N, $D(N, n+h+1)^h > 2^{N-1}$.

(a) Let us assume $N > n+h+1$,

$$\begin{aligned} D(N, n+h+1) &= 2\sum_{i=0}^{n+h}\binom{N-1}{i} \\ &= 2\left(\binom{N-1}{n+h} + \sum_{i=0}^{n+h-1}\binom{N-1}{i}\right) \\ &\leq 2\binom{N-1}{n+h}\{1 + \frac{n+h}{N-n-h-1} + (\frac{n+h}{N-n-h-1})^2 + \ldots\} \\ &= 2\binom{N-1}{n+h}(1 - \frac{n+h}{N-n-h-1})^{-1} \quad \text{since} \quad a + ar + ar^2 + \ldots = \frac{a}{1-r} \\ &= 2\binom{N-1}{n+h}\frac{N-n-h-1}{N-2n-2h-1} \qquad \text{where} \quad |r| < 1 \end{aligned} \tag{D.1}$$

Since $\binom{N}{n+h+1} = \frac{N!}{(n+h+1)!(N-n-h-1)!} = \frac{(N-1)!}{(n+h)!(N-n-h-1)!} \cdot \frac{N}{n+h+1}$, $2\frac{N-n-h-1}{N-2n-2h-1} < \frac{N}{n+h+1}$, and $N \geq (2+\sqrt{2})(n+h+1)+2$, the following subsequently holds

$$2\binom{N-1}{n+h}\frac{N-n-h-1}{N-2n-2h-1} < \frac{N}{n+h+1} \tag{D.2}$$

From equations (D.1) and (D.2) it is easy to conclude that

$$D(N, n+h+1) < \binom{N}{n+h+1} \tag{D.3}$$

For example, let $n+h+1 = 3$ and $N = (2+\sqrt{2})3+2 = 12$. Then $D(12,3) = 2\sum_{i=0} \frac{11!}{i!(11-i)!} = 134 < \binom{12}{3} = 220$.

(b) In order to rewrite $\binom{N}{n+h+1}$. as an expression without any factorials, we use Stirling's approximation in the form $p! \approx \sqrt{2\pi}\cdot p^{p+1/2}\cdot e^{-p+\mu/p}$, where $0 < \mu < 1/12$ [Abramowitz *et al*, 1972]. We proceed by proving

$$\begin{aligned} \binom{N-1}{n+h} &< \{e^{1/12}\cdot(N/(2\pi(n+h+1)(N-n-h-1)))^{1/2}\} \\ &\quad \cdot \frac{N^{n+h+1}}{\{(n+h+1)^{n+h+1}\cdot(1-(n+h+1)/N)^{N-n-h-1}\}} \end{aligned} \tag{D.4}$$

The following approximation and equations hold

$$\binom{N-1}{n+h} = \frac{N!}{(n+h+1)!(N-n-h-1)!}$$

$$
\begin{aligned}
&\approx \frac{\sqrt{2\pi}\cdot N^{N+1/2}\cdot e^{-N+\mu N}}{\sqrt{2\pi}\cdot(n+h+1)^{n+h+1+1/2}\cdot e^{-n-h-1+\mu(n+h+1)}\cdot\sqrt{2\pi}} \\
&\quad\cdot\frac{1}{(N-n-h-1)^{N-n-h-1+1/2}\cdot e^{-N+n+h+1+\mu(N-n-h-1)}} \\
&= \frac{\sqrt{2\pi}\cdot N^N N^{1/2}\cdot e^{-N}e^{\mu N}}{e^{-n-h-1+\mu(n+h+1)-N+n+h+1+\mu(N-n-h-1)}\cdot 2\pi\cdot(n+h+1)^{n+h+1}} \\
&\quad\cdot\frac{1}{(n+h+1)^{1/2}\cdot(N-n-h-1)^N(N-n-h-1)^{-n-h-1}(N-n-h-1)^{1/2}} \\
&= \frac{\sqrt{2\pi}\cdot N^N N^{1/2}\cdot e^{\mu N}}{2\pi\cdot e^{\mu(n+h+1)+\mu(N-n-h-1)}\cdot(n+h+1)^{n+h+1}} \\
&\quad\cdot\frac{1}{(N-n-h-1)^N(N-n-h-1)^{-n-h-1}\cdot((n+h+1)(N-n-h-1))^{1/2}} \\
&= e^{\mu N-\mu(n+h+1)-1/12(N-n-h-1)}\cdot(2\pi)^{-1/2}\cdot N^{N+1/2}\cdot((n+h)(N-n-h-1))^{1/2} \\
&\quad\cdot(n+h+1)^{-n-h-1}\cdot(N-n-h-1)^{-N+n+h+1} \\
&= e^{\mu(N-1/(n+h+1)-1/(N-n-h-1)}\cdot(2\pi(n+h+1)(N-n-h-1))^{-1/2} \\
&\quad\cdot\left(\frac{N}{N-n-h-1}\right)^N\cdot N^{1/2}\cdot\left(\frac{n+h+1}{N-n-h-1}\right)^{-n-h-1} \\
&= e^{\mu\left(\frac{(n+h+1)(N-n-h-1)-N(N-n-h-1)-N(n+h+1)}{N(n+h+1)(N-n-h-1)}\right)}\cdot(N/2\pi(n+h)(N-n-h))^{1/2} \\
&\quad\cdot\left(\frac{1}{1-(n+h)/N}\right)^N\cdot\left(\frac{n+h}{N}\left(\frac{1}{1-(n+h)/N}\right)^{-n-h}\right. \\
&= e^{\mu\left(\frac{(n+h+1)N-N^2-(n+h+1)^2}{N(n+h+1)(N-n-h-1)}\right)}\cdot(N/2\pi(n+h+1)(N-n-h-1))^{1/2} \\
&\quad\cdot\frac{N^{n+h+1}}{(n+h+1)^{n+h+1}}\cdot\left(\frac{1}{1-(n+h)/N}\right)^{N-n-h} \\
&= e^{-\mu\left(\frac{N(n+h+1)+(N-n-h-1)^2}{N(n+h+1)(N-n-h-1)}\right)}\cdot(N/2\pi(n+h+1)(N-n-h-1))^{1/2} \\
&\quad\cdot\frac{N^{n+h+1}}{(n+h+1)^{n+h+1}\cdot(1-(n+h+1)/N)^{N-n-h-1}} \\
&< \{e^{1/12}\ (N/(2\pi(n+h+1)(N-n-h-1)))^{1/2}\} \\
&\quad\cdot N^{n+h+1}/\{(n+h+1)^{n+h+1}\cdot(1-(n+h+1)/N)^{N-n-h-1}\} \qquad \text{(D.5)}
\end{aligned}
$$

The approximate inequality of (D.5) holds, since $e^{-x} < 1$ for $x > 0$ and therefore $e^{1/12} > e^{1/12(-N/(n+h+1)(N-n-h-1))}$. Since $2^{-1/h} > e^{1/12}\cdot(N/(2\pi(n+h+1)(N-n-h-1)))^{1/2}$ and applying inequality (D.5), the following result holds,

$$
\binom{N}{n+h+1} < \{2^{-1/h}\cdot N^{n+h+1}/\{(n+h+1)^{n+h+1}\cdot(1-(n+h+1)/N)^{N-n-h-1}\} \qquad \text{(D.6)}
$$

(c) Since $D(N,n+h+1)^h = 2^{N-1}$ and we have proved that $D(N,n+h+1) < \binom{N}{n+h+1}$,

the following inequalities hold

$$
\begin{aligned}
D(N,n+h+1) &< \{2^{-1/h}\cdot N^{n+h+1}/\{(n+h+1)^{n+h+1}\cdot(1-(n+h+1)/N)^{N-n-h-1}\} \\
D(N,n+h+1)^h &< \{2^{-1}\cdot(N^{n+h+1}/\{(n+h+1)^{n+h+1}\cdot(1-(n+h+1)/N)^{N-n-h-1}\})^h \\
2^N\cdot 2^{-1} &< \{2^{-1}\cdot(N^{n+h+1}/\{(n+h+1)^{n+h+1}\cdot(1-(n+h+1)/N)^{N-n-h-1}\})^h \\
2^{N/h} &< N^{n+h+1}/\{(n+h+1)^{n+h+1}\cdot(1-(n+h+1)/N)^{N-n-h-1}\} \\
(N/h)\log_e 2 &< \log_e\left(\frac{N^{n+h+1}}{\{(n+h+1)^{n+h+1}\cdot(1-(n+h+1)/N)^{N-n-h-1}\}}\right) \\
(N/(n+h+1)h)\log_e 2 &< \log_e\left(\frac{N^{n+h+1}}{\{(n+h+1)^{n+h+1}\cdot(1-(n+h+1)/N)^{N-n-h-1}\}}\right)^{1/(n+h+1)} \\
(N/(n+h+1)h)\log_e 2 &< \log_e(N/\{(n+h+1)\cdot(1-(n+h+1)/N)^{(N-n-h-1)/(n+h+1)}\}) \\
(N/(n+h+1)h)\log_e 2 &< \log_e N-\log_e(n+h+1)-\frac{(N-n-h-1)}{(n+h+1)\log_e(1-(n+h+1)/N)} \\
(N/(n+h+1)h)\log_e 2 &< \log_e(N/(n+h+1)) \\
&\quad -(N/(n+h+1)-1)\log_e(1-1/(N/(n+h+1)) \qquad \text{(D.7)}
\end{aligned}
$$

(d) Let $j = N/(n+h+1)$. Then $\log_e j-(j-1)\log_e(1-1/j) = (j\log_e 2)/h$. For small j the left hand side of this equation is larger. If for example, $h=2$, then the left hand side of the equation is larger for $1<j<10$. Since $-(j-1)\log_e(1-1/j) = (j-1)(1/j+1/(2j^2)+1/(3j^3)+\ldots) = 1-(1-1/2)/j-(1/2-1/3)/j^2-\ldots < 1$, replacing $-(j-1)\log_e(1-1/j)$ by 1 in the equation gives a larger value for the left hand side. The equation can then be rewritten as $ej = eh\log_2(ej)$, since the following equations hold,

$$
\begin{aligned}
\log_e j+1 &= (j\log_e 2)/h \\
\log_e j+\log_e e &= (j\log_e 2)/h \\
\log_e ej &= (j\log_e 2)/h \\
\frac{\log_2 ej}{\log_2 e} &= (j\log_e 2)/h \\
h\log_2(ej) &= j\log_e 2\cdot\log_2 e \\
j &= h\log_2(ej) \\
ej &= eh\log_2(ej) \qquad \text{(D.8)}
\end{aligned}
$$

$ej = eh\log_2(ej)$ has the solution $ej = eh\log_2(eh\log_2(eh\ldots))$, the nesting being infinite [Mitchison *et al*, 1989]. Since $j = N/(n+h+1)$ we can rewrite the last equation (D.8) as $N = (n+h+1)h\log_2(eh\log_2(eh\ldots))$, which leads to $N = O((n+h+1)h\log_2 h)$. The nested expression converges, because the sequence $\{b_n\}$ defined by $b_{n+1} = x\log_2 b_n$ increases and is bounded above by x^2 if $x>4$ [Mitchison *et al*, 1989]. The condition $N \geq (2+\sqrt{2})(n+h+1)+2$

is satisfied, provided that $h \geq 2$. Thus $N = O((n+h+1)h\log_2 h)$ is an upper bound for a solution to $D(N, n+h \div 1)^h = 2^{N-1}$. $\triangle$

D.1.2 n:h:s Upper bound for Elman ASRTNs

Mitchison & Durbin (1989) proved that $O(n(1+h/s)\log_2(1+h/s)$ is an upper bound for the probabilistic capacity of *n:h:s* FFTNs. We now prove, in a similar manner, an upper bound which is an order estimation of the probabilistic capacity of *n:h:s* Elman ASRTNs.

Theorem D.2:

The probabilistic capacity of an Elman ASRTN with n inputs, a single hidden layer of h threshold units, a single context layer of h previous hidden layer values, and s output threshold units is at most $N = O((n+h+1)(1+h/s)\log_2(1+h/s)$ examples in general position, where $h \leq n$.

Proof:

From section 5.1.1.2, it follows that the outputs of the h threshold hidden units associated with each of the N $(n+h)$-dimensional input vectors form a vector with h components, and there are at most $D(N, h+1)$ dichotomies of these N h-dimensional vectors. Each of these dichotomies defines a possible output function for the *n:h:s* Elman ASRTN. Since there are s output threshold units, the number of combinations of output functions are at most $D(N, h+1)^s$. There are therefore at most $D(N, h+1)^s$ ways assigning output functions for each partition defined by the hidden units. Thus the total number of distinct outputs which the *n:h:s* Elman ASRTN can generate is at most $D(N, n+h+1)^h \cdot D(N, h+1)^s$. Since the number of possible outputs for N input vectors is 2^{Ns}, the altered version of $D(N, n+h+1)^h = 2^{N-1}$ (section 5.1.1.2) is [Mitchison *et al*, 1989]

$$D(N, n+h+1)^h \cdot D(N, h+1)^s = 2^{Ns-1} \tag{D.9}$$

Since $h < n+h+1$, we can replace equation (D.9) by $D(N, n+h+1)^h \cdot D(N, n+h+1)^s = 2^{Ns-1}$, which gives $D(N, n+h+1)^{h+s} = 2^{Ns-1}$. Following the same steps as in the proof of Theorem 5.2 (where $N = O((n+h+1)h\log_2 h)$ was derived as an upper bound for the capacity of *n:h:1* Elman ASRTNs), gives the following upper bound for *n:h:s* Elman ASRTNs

$$N \leq (n+h+1)(1+h/s)\log_2(e(1+h/s)\ldots)) = O((n+h+1)(1+h/s)\log_2(1+h/s)) \tag{D.10}$$

where the precondition $h \leq n$ holds. $\triangle$

D.1.3 n:h:1 VC Dimension for Jordan ASRTNs

Sakurai (1993) proved an upper bound of $nh(\log_2 h + 2\log_2(\log_2 h))$ for the VC dimension of *n:h:1* feedforward networks. Using similar steps an corresponding bound for *n:h:1* Jordan ASRTNs is next proved.

Theorem D.3:

The VC dimension of an n:h:1 Jordan ASRTN is at most $(n+1)h(\log_2 h + 2\log_2(\log_2 h))$ when $n+1 \geq 32$ and $h \geq 256$.

Proof:

From definition 5.5, it follows that if it is proven that for any set of N points the number of dichotomies of the set realizable by an *n:h:1* Jordan ASRTN is less than 2^N, N is an upper bound of the VC dimension of such ASRTNs. In the proof of Theorem 5.6 we have deduced that the number of dichotomies of N points in $n+1$ dimensions for h threshold hidden units in an *n:h:1* Jordan ASRTN is $D(N, n+1+1)^h = D(N, n+2)^h$. There are at most $D(N, h+1)$ dichotomies of the N h-dimensional hidden unit vectors, each one of them defining a possible output function for the *n:h:1* Jordan ASRTN. Since there are only one output threshold unit, the number of combinations of output functions are at most $D(N, h+1)^1$. Thus the total number of distinct outputs which the ASRTN can generate is at most $D(N, n+2)^h \cdot D(N, h+1)^1$.

(a) By replacing *n+h+1* with *n+2* in the inequalities (D.3) and (D.6), we obtain for $n+1 \geq 2$ and $N \geq (2+\sqrt{2})(n+1)+2$ the following inequalities

$$\begin{aligned} D(N, n+2) &< \binom{N}{n+2} \\ &< 3 \cdot \binom{N}{n+1} \\ &< 3 \cdot 2^{-1/h} \cdot \frac{N^{n+1}}{(n+1)^{n+1} \cdot (1-(n+1)/N)^{N-n-1}} \\ &< 3 \cdot \frac{N^{n+1}}{(n+1)^{n+1} \cdot (1-(n+1)/N)^{N-n-1}} \\ &< 3 \cdot (e\frac{N}{n+1})^{n+1} \end{aligned} \tag{D.11}$$

since $2^{-1/h} < 1$ and $\frac{1}{(1-(n+1)/N)^{N-n-1}} < e^{n+1}$.

(b) If we set $N = (n+1)h(\log_2 h + 2\log_2(\log_2 h)) = (n+1)h(\log_2 h)(1 + (2\log_2 \log_2 h)/(\log_2 h))$, then the following equations and inequality hold [Sakurai, 1993]

$$\begin{aligned} \log_2 N &= \log_2((n+1)h(\log_2 h)(1 + (2\log_2 \log_2 h)/(\log_2 h)) \\ &= \log_2(n+1)h + \log_2(\log_2 h) + \log_2(1 + (2\log_2 \log_2 h)/(\log_2 h)) \\ &< \log_2(n+1)h + \log_2(\log_2 h) + 1 \end{aligned} \tag{D.12}$$

since $\log_2(1 + (2\log_2 \log_2 h)/(\log_2 h)) < log_2 2 = 1$.

(c) We now only have to prove that $D(N, n+2)^h \cdot D(N, h+1)^1 < 2^N$ or that $\log_2 D(N, n+2)^h \cdot D(N, h+1)^1 - N < 0$. Using the inequalities (D.11) and (D.12), the following equations and inequalities hold

$\log_2 D(N, n+2)^h \cdot D(N, h+1)^1 - N$

$$\begin{aligned} &< \log_2 3(e\frac{N}{(n+1)})^{(n+1)h} \cdot 3(e\frac{N}{h+1})^{(h+1)} - (n+1)h(\log_2 h + 2\log_2 \log_2 h) \\ &= (n+1)h(\log_2 N - \log_2(n+1) + \log_2 e) + h\log_2 3 + \log_2 3 + h(\log_2 N - \log_2 h + \log_2 e) \\ &\quad - (n+1)h(\log_2 h + 2\log_2 \log_2 h) \\ &= (n+1)h\log_2 N - (n+1)h\log_2(n+1) + (n+1)h\log_2 e + h\log_2 3 + \log_2 3 + h\log_2 N \\ &\quad - h\log_2 h + h\log_2 e - (n+1)h\log_2 h - 2(n+1)h\log_2 \log_2 h \\ &< (n+1)h(\log_2(n+1)h + \log_2(\log_2 h) + 1) - (n+1)h\log_2(n+1) + (n+1)h\log_2 e + h\log_2 3 \\ &\quad + \log_2 3 + h(\log_2(n+1)h + \log_2(\log_2 h) + 1) \\ &\quad - h\log_2 h + h\log_2 e - (n+1)h\log_2 h - 2(n+1)h\log_2 \log_2 h \\ &= (n+1)h(1 + \log_2 e + (1/(n+1))\log_2 3 + (1/(n+1))(\log_2(n+1) + \log_2(\log_2 h) + 1 + \log_2 e \\ &\quad + (1/h)\log_2 3) - \log_2 \log_2 h) \\ &< 0 \qquad \text{when } n+1 \geq 2^5 = 32 \text{ and } h \geq 2^8 = 256 \text{ [Sakurai, 1993]} \end{aligned} \tag{D.13}$$

Thus $N = (n+1)h(\log_2 h + 2\log_2(\log_2 h))$ is an upper bound of the VC dimension of *n:h:1* Jordan ASRTNs when $n+1 \geq 32$ and $h \geq 256$. $\triangle$

D.2 Number of hidden units

D.2.1 n:h:1 Lower bound for Jordan ASRTNs

By following similar steps as in the proof of Cosnard *et al* (1992) for the n-dimensional case, we now prove a lower bound for the number of hidden units of an *n:h:1* Jordan ASRTN in terms of δ.

Theorem D.4:

An arbitrary dichotomy of $[0,1]^{n+1}$ can be realized by an *n:h:1* Jordan ASRTN with at least $\lceil \frac{(n+1)}{e}(\lfloor \sqrt{n+1}/\delta \rfloor) \rceil$ hidden threshold units.

Proof:

Consider a set S of all points $x = (x_i)$ in the *(n+1)*-dimensional unit hypercube such that $x_i = j_i(\delta/\sqrt{n+1})$ with $1 \leq i \leq (n+1)$ and j_i an integer (which is the corrected version of Cosnard *et al*'s construction of hyperplanes - see section 5.2). The set S has $N = (\lfloor \sqrt{n+1}/\delta \rfloor + 1)^{n+1}$ elements. We proceed by estimating the number of hyperplanes required to realize the 2^N dichotomies of the Jordan ASRTN. If E is a set of h hyperplanes, then E can realize $2^{C(E)}$ dichotomies, where $C(E)$ is the number of cells defined by E that contain at least one point of S. Since $C(h, n+1)$ is the total number of real cells formed by h hyperplanes in $n+1$ dimensions and $C(h, n+1) = 2^h$ for Jordan ASRTNs (see section 4.3.1), $C(E) \leq C(h, n+1)$ holds. As one hyperplane can only realize a finite number of dichotomies of S, say T unique positions in *(n+1)*-dimensional space, h or fewer hyperplanes cannot realize more than $T^h 2^{C(h,n+1)}$ dichotomies. Therefore, if all dichotomies can be realized by h hyperplanes, $T^h 2^{C(h,n+1)} \geq 2^N$. Cosnard *et al* proved that $C(h,n) \leq 2h^n/n!$ for large h. Accordingly, it follows that $C(h, n+1) \leq \frac{2h^{n+1}}{(n+1)!}$. Since Baum (1988) showed that T is bounded above by a polynomial in the cardinality of S, $T \leq (\lfloor \sqrt{n+1}/\delta \rfloor + 1)^r$ for a given r depending on $n+1$. Therefore $T^h 2^{C(h,n+1)} \geq 2^N$ implies the following equations

$$\begin{aligned}
\log_2 T^h 2^{C(h,n+1)} &\geq \log_2 2^N \\
\log_2 T^h + \log_2 2^{C(h,n+1)} &\geq N \\
\log_2(\lfloor \sqrt{n+1}/\delta \rfloor + 1)^{rh} + C(h, n+1) &\geq (\lfloor \sqrt{n+1}/\delta \rfloor)^{n+1} \\
rh \log_2(\lfloor \sqrt{n+1}/\delta \rfloor + 1) + 2h^{n+1}/(n+1)! &\geq (\lfloor \sqrt{n+1}/\delta \rfloor)^{n+1} \qquad \text{(D.14)}
\end{aligned}$$

For $(n+1) > 1$ and $\lfloor \sqrt{n+1}/\delta \rfloor + 1$ large enough, $rh \log_2(\lfloor \sqrt{n+1}/\delta \rfloor + 1) \geq 2h^{n+1}/(n+1)!$ It then follows from equation (D.14) and from Stirling's approximation to $(n+1)!$, which is $\sqrt{2\pi(n+1)} \cdot (n+1)^{n+1} \cdot e^{-n-1}$, that

$$\begin{aligned}
4h^{n+1}/(n+1)! &\geq (\lfloor \sqrt{n+1}/\delta \rfloor + 1)^{n+1} \\
h &\geq ((n+1)!/4)^{1/(n+1)}(\lfloor \sqrt{n+1}/\delta \rfloor + 1) \\
&= ((n+1)!/4)^{1/(n+1)}(\lfloor \sqrt{n+1}/\delta \rfloor) + ((n+1)!/4)^{1/(n+1)}) \\
&\approx ((n+1)!/4)^{1/(n+1)}(\lfloor \sqrt{n+1}/\delta \rfloor) \\
&\approx ((\sqrt{2\pi(n+1)} \cdot (n+1)^{n+1} \cdot e^{-n-1})/4)^{1/(n+1)}(\lfloor \sqrt{n+1}/\delta \rfloor) \\
&= (\frac{(2\pi(n+1))^{(1/2)(n+1)} \cdot (n+1)}{4^{1/(n+1)} e})(\lfloor \sqrt{n+1}/\delta \rfloor)
\end{aligned}$$

$$\geq \left(\frac{(n+1)}{e}\right)\lfloor\sqrt{n+1}/\delta\rfloor \tag{D.15}$$

Thus $\lceil\frac{(n+1)}{e}\lfloor\sqrt{n+1}/\delta\rfloor\rceil$ is a lower bound for the number of hidden threshold units of an *n:h:1* Jordan ASRTN. $\triangle$

D.2.2 n:h:1 Upper bound for Jordan ASRTNs

Cosnard *et al* (1992) showed that an arbitrary dichotomy (S^+, S^-) can be separated by $n(\lfloor\frac{1}{\delta}\rfloor+1)$ hyperplanes of a feedforward threshold network. In section 5.2 we have showed that this bound should be $n(\lfloor\frac{\sqrt{n}}{\delta}\rfloor+1)$. We next prove in a similar fashion an upper bound for the number of hidden units of an *n:h:1* Jordan ASRTN in terms of the distance between classes.

Theorem D.5:

An arbitrary dichotomy (S^+, S^-) can be separated by $(n+1)(\lfloor\frac{\sqrt{n+1}}{\delta}\rfloor+1)$ hyperplanes of an *n:h:1* Jordan ASRTN.

Proof:

We consider an arbitrary dichotomy (S^+, S^-) in an *(n+1)*-unit hypercube. The hyperplanes of the Jordan ASRTN's hidden layer is constructed in the following manner,
$x_i = b_j, 1 \leq i \leq (n+1), 1 \leq j \leq \lfloor\sqrt{n+1}/\delta\rfloor$, with $b_j = j(\delta/\sqrt{n+1})$ and $(n+1)$ the number of input and state units. The hyperplanes then divide the unit hypercube in $(\lfloor\sqrt{n+1}/\delta\rfloor+1)^{n+1}$ cells [Cosnard *et al*, 1992]. If we assume that no point of the set lies on any of the hyperplanes, then any two points with the maximum distance between them smaller than $\delta/\sqrt{n+1}$ belongs to the same cell and thus to the same class. However, in general some points of the set might lie on some hyperplanes. To account for this case, we translate the hyperplanes slightly toward the origin so that all points belong to the interior of cells by subtracting a value ϵ, with $0 \leq \epsilon \leq \delta/\sqrt{n+1}$, from all the hyperplane equations [Cosnard *et al*, 1992]. Since the distance between two consecutive hyperplanes in each dimension must remain smaller or equal to $\delta/\sqrt{n+1}$, one additional hyperplane for each dimension might be needed. This would result in a total of $\lfloor\sqrt{n+1}/\delta\rfloor+1$ hyperplanes at most per dimension. Since there are $n+1$ dimensions, the number of hyperplanes is $(n+1)(\lfloor\sqrt{n+1}/\delta\rfloor+1)$. $\triangle$

Appendix E

Papers Based on this Dissertation Already Accepted for Publication

- **Cloete, I., & Ludik, J.**, "Increased Complexity Training", in *New Trends in Neural Computation*, eds. Mira, J., Cabestany, J., Prieto, A., in series Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 267-271, 1993.

- **Ludik, J., & Cloete, I.**, "Training Schedules for Improved Convergence", *Proceedings of the International Joint Conference on Neural Networks (IJCNN'93)*, Nagoya, Japan, Vol. I, pp. 561-564, October 1993.

- **Ludik, J., & Cloete, I.**, "Incremental Increased Complexity Training", *Proceedings of the European Symposium on Artificial Neural Networks (ESANN'94)*, Brussels, Belgium, pp. 161-165, 20-21 April 1994.

- **Cloete, I., & Ludik, J.**, "Incremental Training Strategies", *Proceedings of the International Conference on Artificial Neural Networks (ICANN'94)*, Sorrento, Italy, Vol. II, pp. 743-746, 26-29 May 1994.

- **Ludik, J., Van der Poel, E., & Cloete, I.**, "Identification of Finite State Automata in Simple Recurrent Networks", *Proceedings of the World Congress on Neural Networks (WCNN'94)*, San Diego, California, USA, Vol. III, pp. 708-713, 4-9 June 1994.

- **Cloete, I., & Ludik, J.**, "Delta Training Strategies", *Proceedings of the IEEE World Congress on Computational Intelligence (WCCI'94)*, Orlando, Florida, USA, Vol I, pp. 295-298, 26 June - 2 July 1994.

Appendix F

Nomenclature

General:

ABAM	Adaptive Bidirectional Associative Memory
ART	Adaptive Resonance Theory
ASRNN	Architecture-specific Recurrent Neural Network
BP	Backpropagation
BPTT	Backpropagation-Through-Time
FIR	Finite-duration Impulse Response
FP-BPTT	Forward Propagation Backpropagation-Through-Time
H-H	Hidden-to-Hidden
H-O	Hidden-to-Output
ITG	Internal Target Generation
LI	Leaky Integrator
LVQ	Linear Vector Quantization
MDN	Movie Description Network
MLP	Multi-layer Perceptron
O-O	Output-to-Output
RAAM	Recursive Auto-associative Memory
RBP	Recurrent Backpropagation
RFL	Recurrent Flash Learning
RTRL	Real-time Recurrent Learning
STORE	Sustained Temporal Order Recurrent network
TA	Temporal Autoassociation
TAM	Temporal Associative Memory
TBP	Temporal Backpropagation
TCN	Time Concentration Network
TDNN	Time Delay Neural Network
TIM	Time-Intensity-Mapping
TIN	Trainable Inference Network
TPN	Temporal Processing Network
WTA	Winner-Take-All
XOR	Exclusive OR

Training Strategies:

ADST	Alternating Delta Subset Training
ADSET	Alternating Delta Subset Epoch Training
CST	Combined Subset Training
DRM	Delta Ranking Method
DST	Delta Subset Training
FST	Fixed Set Training
FSET	Fixed Set Epoch Training
ICT	Increased Complexity Training
ICET	Increased Complexity Epoch Training
IICT	Incremental Increased Complexity Training
IICET	Incremental Increased Complexity Epoch Training
IL	Incremental Learning
IST	Incremental Subset Training
ISET	Incremental Subset Epoch Training
IADST	Incremental Alternating Delta Subset Training
ILDST	Incremental Largest Delta Subset Training
ISDST	Incremental Smallest Delta Subset Training
IADSET	Incremental Alternating Delta Subset Epoch Training
ILDSET	Incremental Largest Delta Subset Epoch Training
ISDSET	Incremental Smallest Delta Subset Epoch Training
LDST	Largest Delta Subset Training
LDSET	Largest Delta Subset Epoch Training
SDST	Smallest Delta Subset Training
SDSET	Smallest Delta Subset Epoch Training

E_A	Euclidean distance of all the weight changes
isr	Incremental success ratio
l_s	Subset increment for single patterns
l_t	Subset increment for temporal patterns
L	Average length of the temporal patterns
PATtotal	Number of training patterns in fixed set
SUBcurrent	Current subset
RMS	Root Mean Square
RMSdecrement	Value that decrements the RMS termination values linearly
RMSdesired	Desired (user-specified) final RMS error value
RMSstart	RMS error value obtained for the initial subset
RMScurrent	Current RMS error value
S	Number of incremental subsets
S_p	A portion of the incremental subsets
SUBincrement	Increment in subset size

HCA	Hierarchical Cluster Analysis
FSA	Finite State Automata
FSM	Finite State Machine
PCA	Principal Component Analysis
STA	Sammon Transformation Analysis

A	Input alphabet
BR	Begin-Result
C_C	First Carry action in temporal pattern
C_{CC}	Successive Carry actions
CN	Carry-Next
d_{ij}	Euclidian distance between vectors $\vec{y}_i$ and $\vec{y}_j$
d^*_{ij}	Euclidian distance between vectors $\vec{x}_i$ and $\vec{x}_j$
D_C	Done actions preceded by a Result which is due to a Carry
D_N	Done actions preceded by a Next action
D_{CC}	Done actions performed after one Carry and preceded by a Result
D_{CN}	Done actions performed after more than one Carry and preceded by a Next
E	Error value
f	Output function
i	A integer value inclusive between 1 and N
j	A integer value inclusive between 1 and N
L	Dimension of STA input vector
M	Dimension of STA output vector
N	Number of STA input vectors
N_C	Next actions with one Carry earlier in temporal pattern
N_N	Next actions with no Carry actions earlier in temporal pattern
N_{CC}	Next actions performed after more than one Carry and preceded by a Carry
N_{CN}	Next actions performed after more than one Carry and preceded by a Result
ND	Next-Done
NR	Next-Result
O	Output alphabet
R_N	Result actions leading to Next actions
R_C	Result actions leading to Carry actions
R_{CN}	Result actions with earlier Carry actions and lead to Next actions
R_{CC}	Result actions with earlier Carry actions and lead to Carry actions
R_D	Result actions that lead to Done actions (pattern includes one Carry)
R_{CD}	Result actions that lead to Done actions (pattern includes more than one Carry)
RC	Result-Carry
RD	Result-Done
RN	Result-Next
S	A finite set of states
t	time step
T	Transition function
$\vec{x}_i$	STA input vector
$\vec{y}_i$	STA output vector

Complexity Analysis:

ASRTN	Architecture-specific Recurrent Threshold Network
FFTN	Feedforward Threshold Network
VC	Vapnik-Chervonenkis
a_{ij}	A scalar
$C(h,n)$	Number of real cells formed by h hyperplanes in n-space
$C_c(h,n)$	Number of closed cells formed by h hyperplanes in n-space
$C_i(h,n)$	Number of imaginary cells formed by h hyperplanes in n-space
$C_o(h,n)$	Number of open cells formed by h hyperplanes in n-space
$D(N,n)$	Number of dichotomies of N examples in n dimensions
$\vec{e}_i$	A real-valued input vector
f	A $\{0,1\}$-valued function on $\mathcal{R}^n$
E	A set of h hyperplanes
F	A class of $\{0,1\}$-valued functions on $\mathcal{R}^n$
G	The class of functions computed by an Elman or Jordan ASRTN
h	Number of hidden units
h'	An integer number
H	A hidden unit threshold gate
i	An integer number
j	An integer number
m	Sum of input and context (or state) units
n	Number of input units
n'	An integer number
N	Number of input patterns
p	An integer number
p_i	A scalar
P	Minimum number of different kinds of associated output vectors ($\leq 2^s$)
$\mathcal{R}^n$	The set of n-dimensional real numbers
s	Number of output units
$\mathcal{S}$	A set of N points in $\mathcal{R}$
S^+	A finite subset, disjoint from S^-
S^-	A finite subset, disjoint from S^+
T	A finite number of dichotomies
$\vec{u}_i$	A real-valued weight vector
$VC\text{-}dim$	Vapnik Chervonenkis Dimension
$\vec{w}_i$	A real-valued weight vector
w	Number of weights and thresholds
x	A real number
X	A finite subset of S
y	A real number
Y	Smallest integer $\geq$ to the number of n-example sub-subsets per non-background subset
δ	$min\{distance(x,y) \mid x \in S^+, y \in S^-\}$
ϵ	A positive real number
θ_i	A real threshold number
μ	A real number between 0 and 1/12

Index